

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

April 20, 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 7. 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 8 to 10.

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
April 20, 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
April 20, 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
April 20, 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	19
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	41
3.1.2	Non-Empirical Research Questions	41
3.2	Philosophical Stances	42
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	47
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 Better Documenting Computer Vision Services	105
7.1 Introduction	105
7.2 Related Work	108
7.2.1 API Usability and Documentation Knowledge	108
7.2.2 Adapting the System Usability Scale	109
7.2.3 Computer Vision Services	109
7.3 Taxonomy Development	109
7.3.1 Systematic Mapping Study	110
7.3.2 Development of the Taxonomy	114
7.4 API Documentation Knowledge Taxonomy	116
7.5 Validating our Taxonomy	119
7.5.1 Survey Study	119
7.5.2 Empirical application of the taxonomy against Computer Vision Services	120
7.6 Taxonomy Analysis	121
7.6.1 In-Literature Scores for Taxonomy Categories	121
7.6.2 In-Practice Scores for Taxonomy Categories	122
7.6.3 Contrasting In-Literature to In-Practice Scores	123
7.6.4 Triangulating ILS and IPS with Computer Vision	124
7.6.5 Areas of Improvement for CVS Documentation	125
7.7 Threats to Validity	129
7.7.1 Internal Validity	129
7.7.2 External Validity	129
7.7.3 Construct Validity	130
7.8 Conclusions & Future Work	131
8 Using a Facade Pattern to combine Computer Vision Services	133
8.1 Introduction	133
8.1.1 Motivating Scenario: Intelligent vs Traditional Web Services	134
8.1.2 Research Motivation	135
8.2 Merging API Responses	135
8.2.1 API Facade Pattern	136
8.2.2 Merge Operations	136
8.2.3 Merging Operators for Labels	137
8.3 Graph of Labels	137
8.3.1 Labels and synsets	138
8.3.2 Connected Components	138
8.4 API Results Merging Algorithm	141
8.4.1 Mapping Labels to Synsets	141
8.4.2 Deciding Total Number of Labels	141
8.4.3 Allocating Number of Labels to Connected Components . .	141
8.4.4 Selecting Labels from Connected Components	143
8.4.5 Conformance to properties	143
8.5 Evaluation	143
8.5.1 Evaluation Method	143

8.5.2	Naive Operators	144
8.5.3	Traditional Proportional Representation Operators	146
8.5.4	New Proposed Label Merge Technique	146
8.5.5	Performance	146
8.6	Conclusions and Future Work	147
9	Supporting Safe Usage of Intelligent Web Services	149
9.1	Introduction	149
9.2	Motivating Example	152
9.3	Threshy	154
9.4	Related work	155
9.4.1	Decision Boundary Estimation	155
9.4.2	Tooling for ML Frameworks	156
9.5	Conclusions & Future Work	156
10	An Integration Architecture Tactic to Guard AI-first Components	159
10.1	Introduction	159
10.2	Motivating Example	162
10.3	Intelligent Services	163
10.3.1	‘Intelligent’ vs ‘Traditional’ Web Services	163
10.3.2	Dimensions of Evolution	164
10.3.3	Limited Configurability	164
10.4	Our Approach	166
10.4.1	Core Components	167
10.4.2	Usage Example	171
10.5	Evaluation	173
10.5.1	Data Collection and Preparation	173
10.5.2	Results	174
10.5.3	Threats to Validity	175
10.6	Discussion	179
10.6.1	Implications	179
10.6.2	Limitations	180
10.6.3	Future Work	180
10.7	Related Work	181
10.8	Conclusions	181
III	Postface	183
11	Conclusions & Future Work	185
11.1	Contributions of this Work	185
11.1.1	Answers to Research Questions	186
11.1.2	Limitations to Research Answers & Future Research	190
11.2	Concluding Remarks	191
References		213

List of Online Artefacts	215
IV Appendices	219
A Additional Materials	221
A.1 Development, Documentation and Usage of Web APIs	223
A.2 Additional Figures	226
B Reference Architecture Source Code	241
C Supplementary Materials to Chapter 7	275
C.1 Detailed Overview of Our Proposed Taxonomy	277
C.2 Sources of Documentation	281
C.3 List of Primary Sources	284
C.4 Survey Questions	286
D Authorship Statements	291
E Ethics Clearance	327

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: IEEE, October 2020, In Press
4. T. Otake, 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. 3–5, 8, 12–14, 34, 35, 51, 52, 55, 56, 66, 69, 71–73, 75, 76, 86, 89, 90, 92, 93, 135, 136, 159, 163, 175, 181, 188, 190, 191, 226, 228

API application programming interface. xxvi, 4–6, 8–16, 18, 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, 105–110, 112–119, 122–131, 133–136, 138, 140, 141, 143, 144, 147, 150, 159, 160, 163, 166, 168, 173, 179, 181, 185, 187–189, 223, 225

AWS Amazon web services. 57, 60

BYOML Build Your Own Machine Learning. 5, 6

CC connected component. 138, 141–143, 146

CDSS clinical decision support system. 7, 10

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

CRUD create, read, update, and delete. 225

CVS computer vision service. 7–10, 12, 14–21, 23, 25, 27, 28, 31, 37, 40, 41, 43–45, 47, 48, 51–57, 60, 65, 67, 68, 71, 73, 78, 83, 87, 91–93, 95–97, 100, 103, 105–107, 109, 116, 118, 120, 121, 124–131, 133, 135–137, 141, 143, 147, 160, 161, 163, 164, 166, 167, 173, 175, 180, 181, 185–190, 226, 229

DCE distributed computing environment. 223

HITL human-in-the-loop. 11

HTTP Hypertext Transfer Protocol. 6, 170, 171, 174, 179, 223–225

- IDL** interface definition language. 223, 225
- IRR** inter-rater reliability. 91
- IWS** intelligent web service. 5–7, 9–12, 14, 15, 17–19, 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, 131, 133–135, 147, 149, 159, 161–164, 166, 168, 170, 179–182, 185, 190, 191, 226
- JSON** JavaScript Object Notation. 7, 164, 171, 173, 174
- ML** machine learning. 3–6, 8, 9, 12, 13, 18, 22, 29, 34, 37, 51, 52, 55, 56, 66, 68, 72–74, 76, 77, 89, 90, 103, 133, 134, 147, 149
- NN** neural network. 12, 32, 34, 36
- PaaS** Platform as a Service. 7, 11, 55
- QoS** quality of service. 55, 56, 223
- RAML** RESTful API Modeling Language. 225
- REST** REpresentational State Transfer. 5, 52, 71, 92, 134, 159, 181, 224, 225
- ROI** region of interest. 10, 11
- RPC** remote procedure call. 223
- SDK** software development kit. 53, 110, 125
- SLA** service-level agreement. 55, 223
- SMS** systematic mapping study. 18, 20, 22, 107–110, 115, 121, 130, 131
- SO** Stack Overflow. 5, 15, 17, 18, 22, 23, 40, 41, 44, 47, 48, 56, 57, 71, 73–77, 79, 81, 82, 84–87, 89–96, 98–101, 103, 104, 188, 229
- SOA** service-oriented architecture. 223
- SOAP** Simple Object Access Protocol. 5, 223–225
- 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. 18, 106, 107, 109, 119, 120, 122, 130
- SVM** support vector machine. 34, 36

URI uniform resource identifier. 225

V&V verification & validation. 25–29

WADL Web Application Description Language. 225

WSDL Web Services Description Language. 223

XML eXtendable markup language. 7, 223

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	21
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 Systematic mapping study search results, by years	111
7.2 Filtering steps used in the systematic mapping study	111
7.3 A systematic map of API documentation knowledge studies	115
7.4 Our proposed API documentation knowledge taxonomy	117
7.5 Comparison of ILS and IPS values	123
7.6 Comparison of ILS and IPS values	124
8.1 Overview of the proposed facade	136
8.2 Graph of associated synsets against two different endpoints	139
8.3 Label counts per API assessed	140
8.4 Connected components vs. images	140
8.5 Allocation to connected components	142
8.6 F-measure comparison	144
9.1 Example case study of evaluating model performance in two different models	150
9.2 Request and response for computer vision services provide limited configurability	150
9.3 Threshy supports threshold selection and monitoring	152
9.4 Example pipeline of a computer vision system	153
9.5 UI workflow of Threshy	154
9.6 Architecture of Threshy	157
10.1 Prominent computer vision service evolution	160
10.2 Dimensions of evolution within computer vision services	163
10.3 Example of substantial confidence change	164
10.4 Directly versus indirectly accessing intelligent services	165
10.5 Sample request and response for intelligent services	166
10.6 State diagram of architecture workflows	169
10.7 Precondition failure taxonomy	171
10.8 Histogram of confidence variation	174
10.9 Architecture response to substantial confidence evolution	176
10.10 Architecture response to label set evolution	177
10.11 Architecture response of expected label mismatch	178
11.1 Results from computer vision services can be disparate and non-static	187
11.2 Distribution of issues on Stack Overflow	189
A.1 SOAP versus REST search interest over time	224
A.2 Categorisation of AI-based products and services	228
A.3 Increasing interest in the developer community of computer vision services	229
A.4 Causal factors that may influence understanding of intelligent web services	230
A.5 A proposal technical domain model for intelligent services	231

A.6	Potential questions that can be asked around causal factors of a developer's understanding of an intelligent service	232
A.7	Threshy and developer interaction with decision boundaries	232
A.8	Threshy domain model	233
A.9	Threshy sequence diagram	233
A.10	High-level overview of our method in Chapter 5	234
A.11	Class diagram of architecture implementation	235
A.12	Creation of a benchmark using the architecture tactic	236
A.13	Making a request via the proxy server facade	237
A.14	High-level workflow of the architectural tactic	238
A.15	Handling of evolution using our architecture (i)	239
A.16	Handling of evolution using our architecture (ii)	240

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	20
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	Summary of search results in API documentation knowledge	112
7.2	Data extraction in API documentation knowledge study	114
7.3	Weighted ILS Scoring	122
7.4	Weighted IPS Scoring	122
7.5	Labels assigned to ILS and IPS values	123
8.1	Statistics for the number of labels	138
8.2	First allocation iteration	143
8.3	Second allocation iteration	143
8.4	Third allocation iteration	143

8.5	Fourth allocation iteration	143
8.6	Matching to human-verified labels	145
8.7	Evaluation results of the facade	145
8.8	Average of evaluation result of the facade	145
10.1	Potential reasons for precondition failure	167
10.2	Rules encoded within behaviour tokens	168
10.3	Variance in ontologies	175

List of Listings

A.1	An example SOAP request	223
A.2	An example SOAP response	224
A.3	An example RESTful request	225
A.4	An example RESTful response	225
B.1	Implementation of the architecture module components	241
B.2	Implementation of the architecture facade API	267

Part I

Preface

CHAPTER 1

3

4

5

Introduction

6

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.

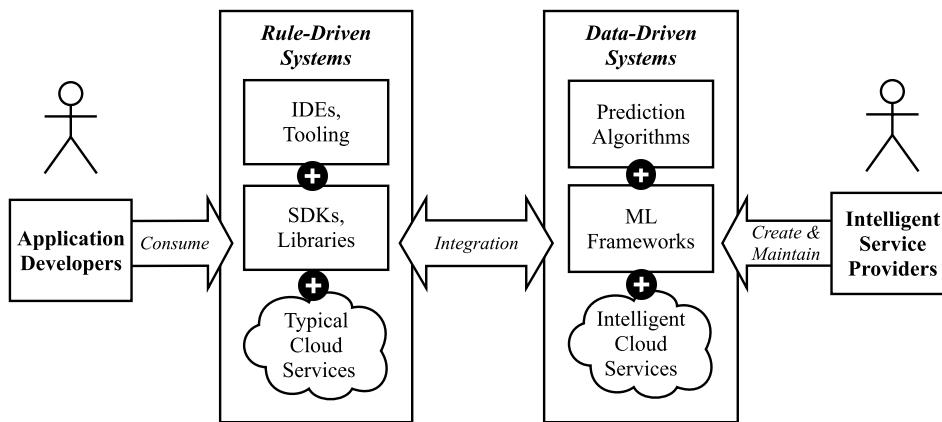
13 However, application developers, accustomed to traditional software engineering
14 paradigms, may not be aware of AI-first’s consequences. Application developers
15 build *rule-driven* applications, where every line of source code evaluates to produce
16 deterministic outcomes. AI-first software is, 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 [205], 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?

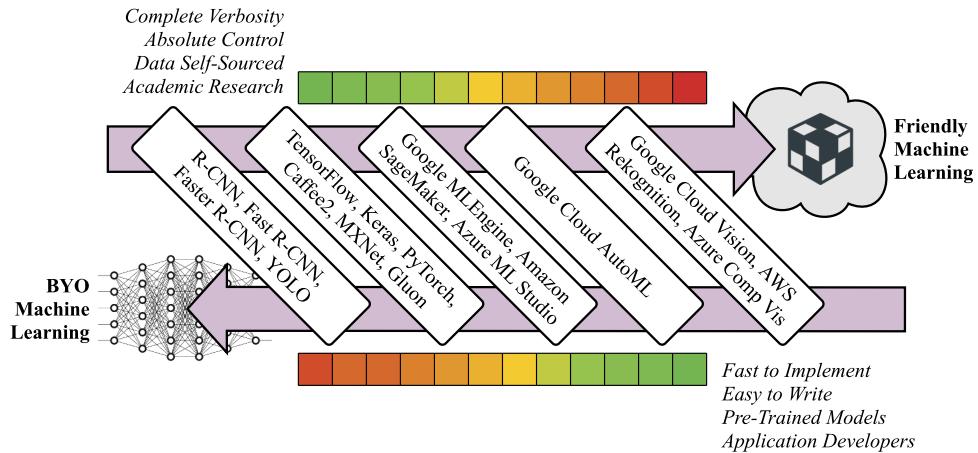
43 Figure 1.1 provides an illustrative overview between the context clashing of rule-
44 driven applications and data-driven cloud services, and we contrast characteristics
45 of typical cloud systems and data-driven ones in Table 1.1.

 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* [190, 217, 248], 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 [166, 217, 248]. 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* [337] or *Machine Learning as a Service* [274]. 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* [151] is evident. A further example includes how Facebook leverage AI *at scale* within their infrastructure and platforms [252].

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

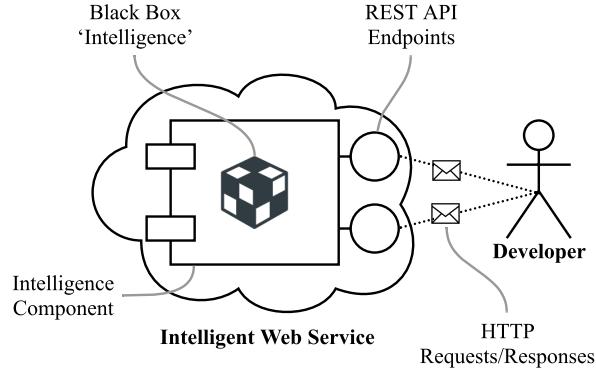
⁵Such as optical character recognition, text-to-speech and speech-to-text transcription, object categorisation, facial analysis and recognition, natural language processing etc.

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	✓	✓			
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 [360, 363, 376, 377, 378, 384, 388, 397, 398, 400, 402, 449, 450]. 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
¹¹⁶ not explicitly documented by the providers (i.e., when the models change) nor do
¹¹⁷ they document what confidence means. Service providers use a tautological nature
¹¹⁸ when defining what the confidence confidence values are (as presented in the API
¹¹⁹ documentation) provides no insight for Tom to understand why there was a change
¹²⁰ in confidence, which we show in Table 1.4, unless he *knows* that the underlying
¹²¹ models change with them. Furthermore, they do not provide detailed understanding
¹²² on how to select a threshold cut-off for a confidence value. Therefore, he's left with
¹²³ no understanding on how best to tune for image classification in this instance. The

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 deterministic problem of a substring compared to the nondeterministic nature of the
¹²⁵ IWS is, therefore, 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).” [386]
	“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].” [387]
	“The overall score of the result. Range [0, 1]” [387]
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.” [403]
Provider C	“The level of confidence the service has in the caption.” [401]
Provider C	“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.” [361]
	“[Provider C] also provide[s] a percentage score for how much confidence [Provider C] has in the accuracy of each detected label.” [362]

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 [27, 136, 204]), 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 [68, 163] 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.

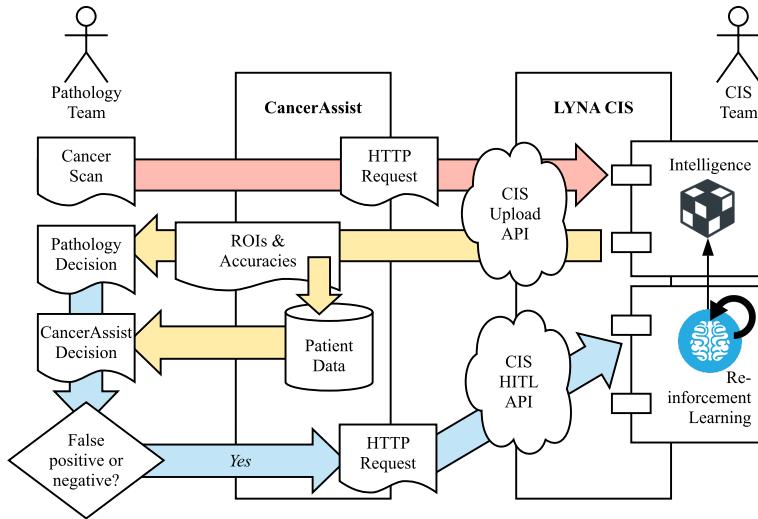


Figure 1.4: CancerAssist Context Diagram. **Key:** Red Arrows = Scan Input; Yellow Arrows = Decision Output; Blue Arrows = HITL Feedback Input.

171 Powering CancerAssist is Google AI’s Lymph Node Assistant (LYNA) [204],
 172 a CNN based on the Inception-v3 model [187, 318]. 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?

- ¹⁹³
- 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 [88, 272], accounting [211], data analytics [161], and ²⁰⁰ student education [94]. 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 [258] 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 [12, 58] to describe “systems whose internal mechanisms are not fully open to inspection” [12].

¹⁰Blake et al. [38] produces a multi-layer perceptron neural network performing arithmetic representation.

228 APIs when the *internal model* changes [81]. 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.” [132]. 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 [263, 279] and
267 providers should consider mismatches between a developer’s conceptual knowledge
268 of the API its implementation [182]. **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 [215] and Boehm [40] 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 [225], 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 7**). 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.

② 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 We formulate the following RQs:

**③ 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?

318 Lastly, we explore several strategies to help improve CVSs reliability. Firstly,
319 we investigate whether merging the responses of *multiple* CVSs can improve their
320 reliability and propose a novel algorithm—based on the proportional representation
321 method used in electoral systems—to merge labels and associated confidence values
322 from three providers (**Chapter 8**). Secondly, we develop an integration architecture
323 style (or facade) to guard against CVS evolution, and synthesise an integration
324 workflow that addresses the concerns raised by developers in addition to embedding
325 ‘*complete*’ documentation artefacts into the workflow’s design (**Chapters 9 and 10**).
326 Our final RQ is:

**④ RQ4. What strategies can developers employ to integrate their appli-
cations with CVSs while preserving robustness and reliability?**

327 1.5 Research Methodology

328 This thesis employs a mixed-methods approach using the concurrent triangulation
329 strategy [51, 214]. The research presented consists of both empirical and non-
330 empirical research design. This section provides a high-level overview of the re-
331 search methodology within this thesis. Further details are provided in Section 1.7
332 and Chapter 3.

333 Firstly, RQ1–RQ3 are all empirical, knowledge-based questions [103, 221] that
334 aim to provide the software engineering community with a greater understanding
335 of the phenomena surrounding CVSs from three perspectives: the nature of the ser-
336 vices themselves, how developers perceive these services and how service providers
337 can improve these services. We answer RQ1 using a longitudinal experiment that
338 assesses both the services’ responses and associated documentation (complement-
339 ing RQ2.2). We adopt qualitative and quantitative data collection; specifically (i)
340 structured observations to quantitatively analyse the results over time, and (ii) docu-
341 mentary research methods to inspect service documentation. Secondly, we perform
342 systematic mapping study following the guidelines of Kitchenham and Charters
343 [178] and Petersen et al. [260] to better understand how API documentation of these
344 services can be improved (i.e., more complete), which targets RQ2. Based on the
345 findings from this study, we use a systematic taxonomy development methodology
346 specifically targeted toward software engineering [330] that structures scattered API
347 documentation knowledge into a taxonomy. We then validate this taxonomy against
348 practitioners using survey research, adopting Brooke well-established Systematic
349 Usability Score [55] surveying instrument and contextualising it within API docu-
350 mentation utility, which answers RQ3.3. To answer RQ2.2, we perform an empirical
351 application of the taxonomy to three CVSs, and therefore assess where improve-
352 ments can be made. Thirdly, we adopt field survey research using repository mining
353 of developer discussion forums (i.e., Stack Overflow) to answer RQ3, and classify
354 these using both manual and automated techniques.

355 The second aspect of our research design involves non-empirical research, which
356 explores a design-based question [225] to answer RQ4. As the answers to our
357 first three RQs establish a greater understanding of the nature behind CVSs from
358 various perspectives, the strategies we design in RQ4 aims at designing more reliable
359 integration methods so that developers can better use these cloud-based services in
360 their applications.

361 1.6 Thesis Organisation

362 We organise the thesis into four parts. **Part I (The Preface)** includes introduc-
363 tory, background and methodology chapters. This is a *PhD by Publication*, and
364 **Part II (Publications)** comprises of seven publications resulting from this work
365 over Chapters 4 to 10; publications are included verbatim except for terminology
366 and formatting changes to better fit the suitability of a coherent thesis. **Part III (The**
367 **Postface)** includes the conclusion and future works chapter, as well as a list of aca-
368 demic studies and online artefacts referenced within the thesis. **Part IV (Appendices)**

369 includes all supplementary material, including mandatory authorship statements and
370 ethics approval. Details of each chapter following this introductory chapter are pro-
371 vided in the following section.

372 1.6.1 Part I: Preface

373 1.6.1.1 *Chapter 2: Background*

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

379 1.6.1.2 *Chapter 3: Research Methodology*

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

384 1.6.2 Part II: Publications

385 1.6.2.1 *Chapter 4: Exploring the nature of CVSSs*

386 This chapter was presented at the 2019 International Conference on Software
387 Maintenance and Evolution (ICSME) [81]. We describe an 11-month longitudinal
388 experiment assessing the behavioural (run-time) issues of three popular CVSSs:
389 Google Cloud Vision [388], Amazon Rekognition [363] and Azure Computer Vi-
390 sion [402]. By using three different data sets—two of which we curate as additional
391 contributions—we demonstrate how the services are inconsistent amongst each other
392 and within themselves. This study provides a detailed answer to RQ1: Despite
393 presenting conceptually-similar functionality, each service behaves and produces
394 slightly varied (inconsistent) results and demonstrates non-deterministic runtime
395 behaviour. We discuss potential evolution risks to consumers of such services as the
396 services provide non-static outputs for the same inputs, thereby having significant
397 impact to the robustness of consuming applications. Further details in the study
398 include a brief assessment into the lack of sufficient detail of these concerns in their
399 documentation.

400 1.6.2.2 *Chapter 5: Understanding developer struggles when using CVSSs*

401 This chapter has been accepted for presentation at the 2020 International Conference
402 on Software Engineering (ICSE) [84]. We conduct a mining study of 1,425 Stack
403 Overflow questions that provide indications of the types frustrations that developers
404 face when integrating CVSSs into their applications. To gather what their pain-
405 points are, we use two classification taxonomies that also use Stack Overflow to

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

419 *1.6.2.3 Chapter 6: Ranking CVS pain-points by frustration*

420 This chapter has been published as a technical report pre-print on arXiv and an
421 extended version is in progress for submission to IEEE Software [86]. In this work,
422 we use our dataset consisting of the 1,425 Stack Overflow (SO) questions from [84]
423 to interpret the breakdown of emotions developers express per classification of pain-
424 points conducted in Chapter 5. We find that the distribution of various emotions
425 differ per question type, and developers are most frustrated when the expectations
426 of a CVS does not match the reality of what these services actually provide, which
427 shapes our answer for RQ3.2 and thus RQ3.

428 *1.6.2.4 Chapter 7: Investigating improvements to CVS API documentation*

429 This chapter was originally a short paper presented at the 2019 International Sym-
430 posium on Empirical Software Engineering and Measurement (ESEM) [84]. To
431 understand where to improve CVS documentation, we first need to investigate *what*
432 makes a good API document. This short paper initially answered one aspect of
433 RQ2.1: what *academic literature* suggests a good (complete) API document should
434 comprise of. By conducting an systematic mapping study resulting in 21 primary
435 studies, we systematically develop a taxonomy that combines the recommendations
436 of scattered work into a structured framework of 5 dimensions and 34 weighted cat-
437 egorisations. We then extend this work by triangulating the taxonomy with opinions
438 from developers using the System Usability Scale to assess the efficacy of these
439 recommendations (thereby answering the second aspect of RQ2.1). From this, we
440 assess the how well CVS providers document their APIs via a heuristic validation
441 of the taxonomy, using the three services from the ICSME publication to make rec-
442 ommendations where documentation should be more complete, thereby answering
443 RQ2.2 (and thus RQ2). The extended version of this chapter has been submitted to
444 the IEEE Transactions on Software Engineering (TSE) in [85] and is currently in
445 review.

446 *1.6.2.5 Chapter 8: Merging responses of multiple CVSs*

447 This chapter was presented at the 2019 International Conference on Web Engineering
448 (ICWE) [245]. Early exploration of CVSs showed that multiple services use
449 vastly different ontologies for the same input. As an initial strategy to improve
450 the reliability of these services, we explored if merging multiple responses using
451 WordNet [227] and a novel label merging algorithm based on the proportional rep-
452 resentation approach used in political voting could make any improvements. While
453 this approach resulted in a modest improvement to reliability, it did not consider to
454 the evolution issues or developer pain-points we later identified.

455 *1.6.2.6 Chapter 9: Developing a confidence thresholding tool*

456 This chapter has been submitted to the demonstrations track at FSE 2020 [82]. When
457 integrating with a CVS, developers need to select an appropriate confidence threshold
458 suited to their use case and determine whether a decision should be made. An issue,
459 however, is that these CVSs are not calibrated to the specific problem-domain datasets
460 and it is difficult for software developers to determine an appropriate confidence
461 threshold on their problem domain. This tool presents a workflow and supporting
462 tool for application developers to select decision thresholds suited to their domain
463 that—unlike existing tooling—is designed to be used in pre-development, pre-release
464 and production. This tooling forms part of a solution to RQ4 for developers to
465 maintain robustness and reliability in their systems.

466 *1.6.2.7 Chapter 10: Developing a CVS integration architecture*

467 This chapter has been submitted to the 2020 Joint European Software Engineer-
468 ing Conference and Symposium on the Foundations of Software Engineering [83].
469 *⟨ TODO: AC: Added findings from this paper Based on the findings, we propose a set of
470 new service error codes for describing the empirically observed error conditions of
471 IWS based on our findings in Chapter 4. To achieve this, we propose a proxy server
472 intermediary that lies between a client application and a IWS; the proxy server tactic
473 is designed to return these error codes when substantial evolution occurs against a
474 benchmark dataset that represents the application domain context (similar to that
475 proposed in Chapter 9). A technical evaluation of our implementation of this archi-
476 tecture identifies 1,054 cases of substantial evolution in confidence values and 2,461
477 cases of evolution in the response label sets when 331 images were sent to a CVS.⟩*

478 **1.6.3 Part III: Postface**

479 In Chapter 11, we review the contributions made in this thesis and the relevance
480 and significance to identifying and resolving key issues when application developers
481 integrate with CVS. We evaluate these outcomes with reference to the research goals,
482 and discuss threats to validity of the work. Lastly, we discuss the various avenues
483 of research arising from this work. References from literature and a list of online
484 artefacts are provided after this concluding chapter.

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
[81]	35 th International Conference on Software Maintenance and Evolution	ICSME	A	05 Dec 2019	Chapter 4	RQ1
[80]	13 th International Symposium on Empirical Software Engineering and Measurement	ESEM	A	17 Oct 2019	Excluded ¹⁵	RQ2.1
[84]	42 nd International Conference on Software Engineering	ICSE	A*	In Press	Chapter 5	RQ3
[86]	IEEE Software	–	Q2	In Progress	Chapter 6	RQ3.2
[85]	IEEE Transactions on Software Engineering	TSE	Q1	In Review	Chapter 7	RQ2
[245]	13 th International Conference on Web Engineering	ICWE	B	26 Apr 2019	Chapter 8	RQ4
[82]	28 th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	FSE(d) ¹⁶	A*	In Review	Chapter 9	RQ4
[83]	28 th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	FSE	A*	In Review	Chapter 10	RQ4

1.6.4 Part IV: Appendices

⁴⁸⁵ Appendix A provides additional material referenced within this thesis but not provided in the body. The source code for the reference architecture described in Chapter 10 is reproduced in Appendix B. The supplementary materials published with Chapter 7 are reproduced in Appendix C, which also describes the list of primary sources arising in the systematic mapping study we conducted. We provide mandatory coauthor declaration forms describing the contribution breakdown for each publication within Appendix D. Appendix E contains copies of the ethics clearance for various experiments within this thesis.

1.7 Research Contributions

⁴⁹⁴ The outcomes of answering the four primary research questions elaborated in Section 1.4 shapes three primary contributions this thesis offers to software engineering knowledge:

- ⁴⁹⁸ • An **improved understanding in the landscape of CVSSs**, with respect to their runtime behaviour and evolutionary profiles.

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 7.

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

- 500 • A novel **service integration architecture** that helps developers with integrating
 501 their applications with CVSs.
 502 • A **key list of attributes that should be documented**, to assist CVS providers
 503 to better document their services.

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

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

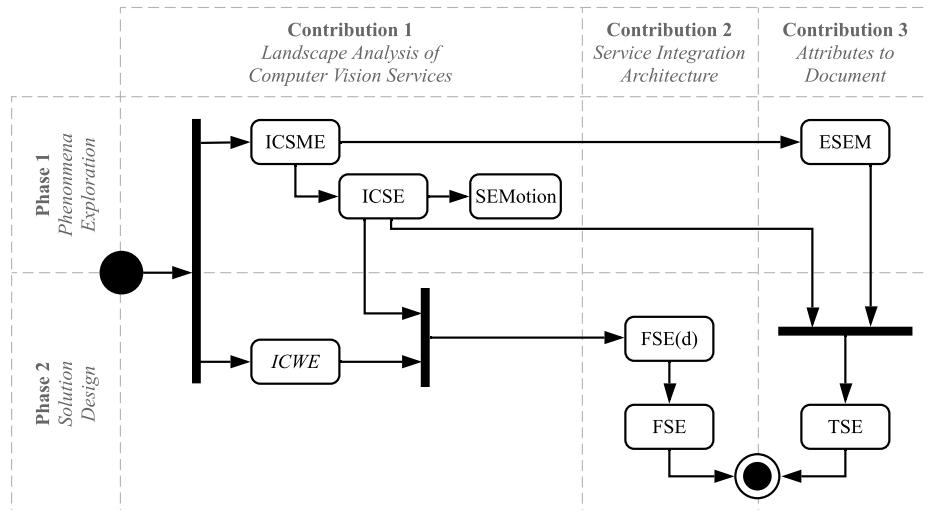


Figure 1.5: Activity diagram of the coherency of our publications, how our research was conducted, and relevant connections between publications. Our two-phase structure initial phenomena exploration and a proposed solutions to issues identified from the exploration. We map the contributions within each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

517 1.7.1 Contribution 1: Landscape Analysis & Preliminary Solutions

518 The first two bodies of work in this paper are the ICSME and ICWE papers. These
 519 two works investigated a landscape analysis CVSs from two perspectives: firstly, we
 520 conducted a longitudinal study to better understand the attributes associated with

521 these services (ICSME)—particularly their evolution and behavioural profiles, and
522 their potential impacts to software reliability—and tackled a preliminary solution
523 facade to ‘merge’ responses of the services together (ICWE).

524 The ICSME paper confirmed our hypotheses that the services have a non-
525 deterministic behavioural profile, and that the evolution occurring within the ML
526 models powering these services are not sufficiently communicated to software en-
527 gineers. This therefore led to follow up investigation into how developers perceive
528 these services, and thereby determine if they are frustrated due to this lack of com-
529 munication.

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

544 1.7.2 Contribution 2: Improving Documentation Attributes

545 As mentioned, our ICSME paper found that evolutionary and non-deterministic
546 behavioural profile of are not adequately documented in the service’s APIs docu-
547 mentation. A recommendation concluding from this work was that service providers
548 should improve their documentation, however there lacked a strategy by which they
549 could do this, and our hypotheses that developers were actually frustrated by this
550 lack of communication was yet to be tested. This led to two follow-up further
551 investigations as presented in our ICSE and ESEM papers.

552 One aspect of our ICSE paper was to confirm whether developers are actually
553 frustrated with the service’s limited API documentation. By mining Stack Overflow
554 posts with reference to documentation issues, we adopted a 2019 documentation-
555 related taxonomy by Aghajani et al. [2] to classify posts, and found that 47.87%
556 of posts classified fell under the ‘completeness’ dimension of Aghajani et al.’s
557 taxonomy. This interpretation, therefore, warranted the recommendation proposed
558 in the ICSME paper to improve service documentation.

559 However, though improvements to more complete documentation was justified
560 from the ICSE paper, we needed to explore exactly *what* makes a ‘complete’ API
561 document. By conducting a systematic mapping study resulting in 4,501 results, we
562 curated 21 primary studies that outline the facets of API documentation knowledge.
563 From these studies, we distilled a documentation framework describing a priori-

564 tised order of the documentation assets API’s should document that is described
565 in our ESEM short paper. After receiving community feedback, we extended this
566 short paper with a follow-up experiment submitted to TSE. By conducting a sur-
567 vey with developers, we assessed our API documentation taxonomy’s efficacy with
568 practitioner opinions, thereby producing a weighted taxonomy against *both* literature
569 and developer sources. Lastly, we triangulated both weightings against a heuristic
570 evaluation against common CVS providers’ documentation. This allowed us to de-
571 duce which specific areas in existing CVS providers’ API documentation needed
572 improvement, which was a primary contribution from our TSE article.

573 1.7.3 Contribution 3: Service Integration Architecture

574 Two recommendations from our ICSME study encouraged developers to test their
575 applications with a representative ontology for their problem domain and to incorpo-
576 rate a specialised testing and monitoring techniques into their workflow. Strategies
577 on *how* to achieve this were explored in later studies. Following a similar approach
578 to our solution of improved API documentation, we validated the substantiveness of
579 our recommendations using our mining study of Stack Overflow (our ICSE paper)
580 to help inform us of generalised issues developers face whilst integrating CVSs into
581 their applications. To achieve this, we used a Stack Overflow post classification tax-
582 onomy proposed by Beyer et al. [34] into seven categories, where 28.9% and 20.37%
583 of posts asked issues regarding how to use the CVS API and conceptual issues be-
584 hind CVSs, respectively. Developers presented an insufficient understanding of the
585 non-deterministic runtime behaviour, functional capability, and limitations of these
586 services and are not aware of key computer vision terminology. When contrasted
587 to more conventional domains such as mobile-app development, the spread of these
588 issues vary substantially.

589 We proposed two technical solutions in our two FSE papers to help alleviate
590 this issue. Firstly, our FSE demonstrations paper—FSE(d) for short—provides a
591 workflow for developers to better select an appropriate confidence threshold, and
592 thus decision boundary, calibrated for their particular use case. In our ESEC/FSE
593 paper, we provide a reference architecture for developers to guard against the non-
594 deterministic issues that may ‘leak’ into their applications. This architecture tactic
595 proposes a client-server intermediary proxy server, similar to the style proposed in
596 our ICWE paper. However, unlike the ICWE paper that uses proportional repre-
597 sentation approach to modify multiple sources, our FSE paper proposes a guarded
598 relay, whereby a single service is used, and the proxy server maintains a lifecycle to
599 monitor evolution issues identified in ICSME and should be benchmarked against
600 the developer’s dataset (i.e., against the particular application domain) as suggested
601 in FSE(d). *< todo: AC: Revised this text. For robust component composition, this*
602 *architecture tactic handles four key requirements: (i) it clearly defines erroneous*
603 *conditions that occur when evolution occurs in CVSs; (ii) it notifies of behavioural*
604 *changes in the service; (iii) it monitors the service for change and substantial impact*
605 *this may have to the client application; and (iv) is flexible enough to be implemented*
606 *and adaptable to any client application or specific intelligent service to facilitate*

⁶⁰⁷ reuse. Both FSE papers serve as two primary contributions to RQ4. ›

CHAPTER 2

608

609

610

Background

611

612 In Chapter 1, we defined a common set of (artificial) intelligence-based cloud ser-
613 vices that we label intelligent web services (IWSs). Specifically, we scope the
614 primary body of this study’s work on computer vision services (CVSs) (e.g., Google
615 Cloud Vision [388], AWS Rekognition [363], Azure Computer Vision [402], Watson
616 Visual Recognition [398] etc.). We claim developers have a distinctly determinis-
617 tic mindset ($2 + 2$ always equals 4) whereas an IWS’s ‘intelligence’ component (a
618 black box) may return probabilistic results ($2 + 2$ might equal 4 with a confidence
619 of 95%). Thus, there is a mindset mismatch between probabilistic results (from the
620 API provider) and results interpreted with certainty (from the API consumer).

621 What affect does this mindset mismatch have on the developer’s approach to-
622 wards building probabilistic software? What can we learn from common software
623 engineering practices (e.g., [266, 309]) that apply to resolve this mismatch and
624 thereby improve quality, such as verification & validation (V&V)? Chiefly, we an-
625 chor this question around three lenses of software engineering: creating an IWS,
626 using an IWS, and the nature of IWSs themselves.

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

- 631 **(1) The development of the IWS.** We investigate the internal quality attributes
632 of creating an IWS from the IWS *provider’s* perspective. That is, we ask if
633 existing verification techniques are sufficient enough to ensure that the IWS
634 being developed actually satisfies the IWS consumer’s needs and if the internal
635 perspective of creating the system with a non-deterministic mindset clashes
636 with the outside perspective (i.e., pillar 2).
- 637 **(2) The usage of the IWS.** We investigate the external quality attributes of using
638 an IWS from the IWS *consumer’s* perspective. That is, we ask if existing
639 validation techniques are sufficient enough to ensure that the end-users can

640 actually use an IWS to build their software in the ways they expect the IWS to
 641 work.

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

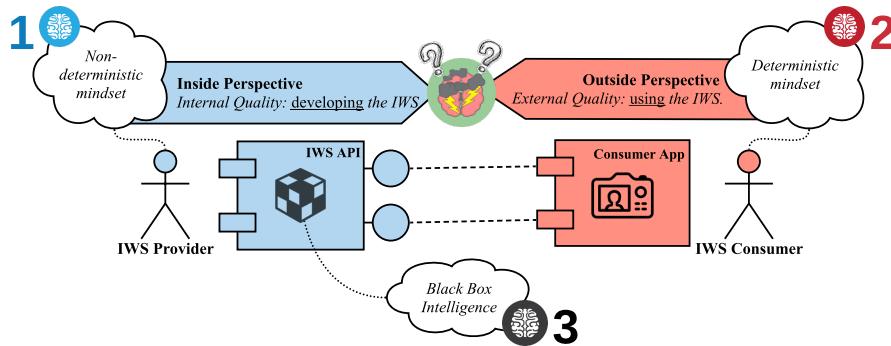


Figure 2.1: The three pillars by which we anchor the background: (1) developing an IWS with a non-deterministic mindset by the IWS provider; (2) the use of a IWS with a deterministic mindset by the IWS consumer; (3) the nature of a IWS itself.

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

652 2.1 Software Quality

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

ROBERT PIRSIG, 1974 [264]

654 The philosophical viewpoint of ‘quality’ remains highly debated and there are mul-
 655 tiple facets to perceive this complex concept [123]. Transcendentally, a viewpoint
 656 like that of Pirsig’s above shows that quality is not tangible but still recognisable; it’s
 657 hard to explicitly define but you know when it’s missing. The International Orga-
 658 nization for Standardization provides a breakdown of seven universally-applicable
 659 principles that defines quality for organisations, developers, customers and training
 660 providers [158]. More pertinently, the 1986 ISO standard for quality was simply
 661 “the totality of characteristics of an entity that bear on its ability to satisfy stated or
 662 implied needs” [157].

Using this sentence, what characteristics exist for non-deterministic IWSs like that of a CVS? How do we know when the system has satisfied its ‘stated or implied needs’ when the system can only give us uncertain probabilities in its outputs? Such answers can be derived from related definitions—such as ‘conformance to specification or requirements’ [79, 128], ‘meeting or exceeding customer expectation’ [31], or ‘fitness for use’ [170]—but these then still depend on the solution description or requirements specification, and thus the same questions still apply.

Software quality is somewhat more concrete. Pressman [266] adapted the manufacturing-oriented view of quality from [32] and phrased software quality under three core pillars:

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

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

2.1.1 Validation and Verification

To explain V&V, we analogously recount a tale given by Pham [262] on his works on reliability. A high-school student sat a standardised test that was sent to 350,0000 students [319]. A multiple-choice algebraic equation problem used a variable, a , and intended that students *assume* that the variable was non-negative. Without making this assumption explicit, there were two correct answers to the multiple choice answer. Up to 45,000 students had their scores retrospectively boosted by up to 30 points for those who ‘incorrectly’ answered, however, outcomes of a student’s higher education were, thereby, affected by this one oversight in quality assessment. The examiners wrote a poor question due to poor process standards to check if their ‘correct’ answers were actually correct. The examiners “didn’t build the right product” nor did they “build the product right” by writing an poor question and failing to ensure quality standards, in the phrases Boehm [42] coined.

706 This story describes the issues with the cost of quality [41] and the importance
707 of V&V: just as the poorly written exam question had such a high toll the 45,000
708 unlucky students, so does poorly written software in production. As summarised by
709 Pressman [266], data sourced from Digital [73] in a large-scale application showed
710 that the difference in cost to fix a bug in development versus system testing is
711 \$6,159 per error. In safety-critical systems, such as self-driving cars or clinical
712 decision support systems, this cost skyrockets due to the extreme discipline needed
713 to minimise error [322].

714 Formally, we refer to the IEEE Standard Glossary of Software Engineering
715 Terminology [154] for to define V&V:

716 verification	The process of evaluating a system or component to determine 717 whether the products of a given development phase satisfy the 718 conditions imposed at the start of that phase.
719 validation	The process of evaluating a system or component during or at the 720 end of the development process to determine whether it satisfies 721 specified requirements.

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

724 The verification process of API providers ‘leak’ out to the context of the de-
725 veloper’s project dependent on the IWS. Poor verification in the *internal quality*
726 of the IWS will entail poor process standards, such as poor definitions and termi-
727 nology used, support tooling and description of documentations [309]. Though
728 it is commonplace for providers to have a ‘ship-first-fix-later’ mentality of ‘good-
729 enough’ software [333], the consequence of doing so leads to consumers absorb-
730 ing the cost. Thus API providers must ensure that their verification strategies
731 are rigorous enough for the consumers in the myriad contexts they wish to use
732 it in. Studies have considered V&V in the context of web services on the cloud
733 [16, 63, 64, 111, 143, 235, 237, 353], though little have recently considered how
734 adding ‘intelligence’ to these services affects existing proposed frameworks and
735 solutions. For a CVS, what might this entail? Which assurances are given to the
736 consumers, and how is that information communicated? To verify if the service is
737 working correctly, does that mean that we need to deploy the system first to get a
738 wider range of data, given the stochastic nature of the black box?

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

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

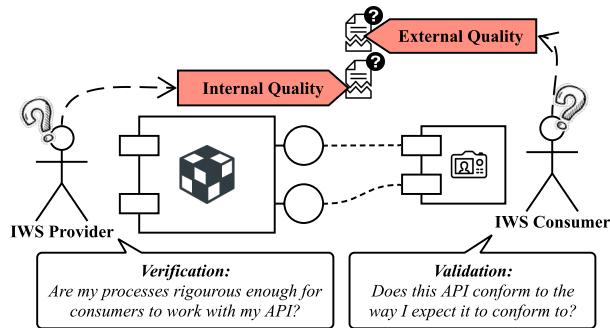


Figure 2.2: The ‘leakage’ of internal quality into the API consumer’s product and external quality imposing on the API provider.

749 certain. While V&V has been investigated in the area of mathematical and earth
 750 sciences for numerical probabilistic models and natural systems [247, 288], from
 751 the software engineering literature, little work has been achieved to look at the
 752 surrounding area of probabilistic systems hidden behind API calls.

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

765 2.1.2 Quality Attributes and Models

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

770 Quality attribute assessment models (like those shown in Figure 2.3) are an early
 771 concept in software engineering, and systematically evaluating software quality
 772 appears as early as 1968 [287]. Rubey and Hartwick’s 1968 study introduced the
 773 phrase ‘attributes’ as a “prose expression of the particular quality of desired software”
 774 (as worded by Boehm et al. [40]) and ‘metrics’ as mathematical parameters on a
 775 scale of 0 to 100. Early attempts to categorise wider factors under a framework was
 776 proposed by McCall, Richards, and Walters in the late 1970s [67, 215]. This model
 777 described quality from the three perspectives of product revision (*how can we keep*

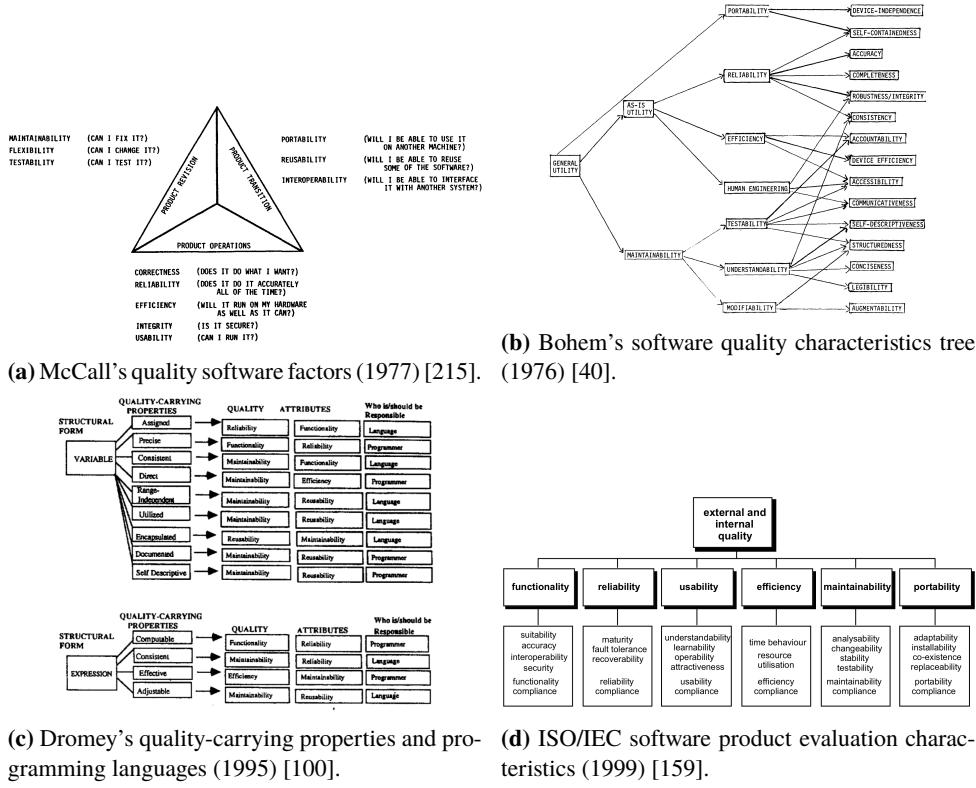


Figure 2.3: A brief overview of the development of software quality models since 1977.

778 *the system operational?), transition (how can we migrate the system as needed?)*
 779 *and operation (how effective is the system at achieving its tasks?)* (Figure 2.3a).
 780 The model also introduced 11 attributes alongside numerous direct and indirect
 781 measures to help quantify quality. This model was further developed by Boehm
 782 et al. [40] who independently developed a similar model, starting with an initial set
 783 of 11 software characteristics. It further defined candidate measurements of Fortran
 784 code to such characteristics, taking shape in a tree-like structure as in Figure 2.3b.
 785 In the mid-1990s, Dromey's interpretation [100] defined a set of quality-carrying
 786 properties with structural forms associated to specific programming languages and
 787 conventions (Figure 2.3c). The model also supported quality defect identification
 788 and proposed an improved auditing method to automate defect detection for code
 789 editors in IDEs. As the need for quality models became prevalent, the International
 790 Organization for Standardization standardised software quality under ISO/IEC-9126
 791 [159] (the Software Product Evaluation Characteristics, Figure 2.3d), which has since
 792 recently been revised to ISO/IEC-25010 with the introduction of the Systems and
 793 software Quality Requirements and Evaluation (SQuaRE) model [156], separating
 794 quality into *Product Quality* (consisting of eight quality characteristics and 31 sub-
 795 characteristics) and *Quality In Use* (consisting of five quality characteristics and 9
 796 sub-characteristics). An extensive review on the development of quality models in
 797 software engineering is given in [5].

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

- 805 **McCall et al.** Extent to which a program can be expected to perform its in-
 806 tended function with required precision [215].
- 807 **Boehm et al.** Code possesses the characteristic *reliability* to the extent that
 808 it can be expected to perform its intended functions satisfac-
 809 torily [40].
- 810 **Dromey** Functionality implies reliability. The reliability of software is
 811 therefore dependent on the same properties as functionality, that
 812 is, the correctness properties of a program [100].
- 813 **ISO/IEC-9126** The capability of the software product to maintain a specified
 814 level of performance when used under specified conditions [159].

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

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

829 2.1.3 Reliability in Computer Vision

830 Testing computer vision deep-learning reliability is an area explored typically
 831 through the use of adversarial examples [317]. These input examples are where
 832 images are slightly perturbed to maximise prediction error but are still interpretable
 833 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.

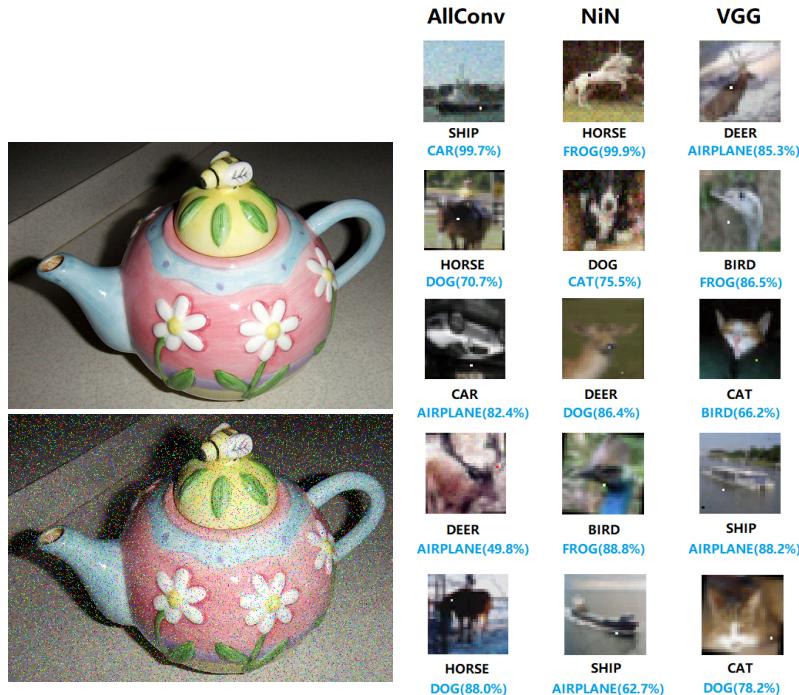


Figure 2.4: Sample adversarial examples in state-of-the-art computer vision studies.

834 Google Cloud Vision, for instance, fails to correctly classify adversarial examples
 835 when noise is added to the original images [149]. Rosenfeld et al. [285] illustrated
 836 that inserting synthetic foreign objects to input images (e.g., a cartoon elephant)
 837 can alter classification output. Wang et al. [336] performed similar attacks on a
 838 transfer-learning approach of facial recognition by modifying pixels of a celebrity's
 839 face to be recognised as a different celebrity, all while still retaining the same human-
 840 interpretable original celebrity. Su et al. [312] used the ImageNet database to show
 841 that 41.22% of images drop in confidence when just a *single pixel* is changed in the
 842 input image; and similarly, Eykholt et al. [106] recently showed similar results that
 843 made a CNN interpret a stop road-sign (with mimicked graffiti) as a 45mph speed
 844 limit sign.

845 Thus, the state-of-the-art computer vision techniques may not be reliable enough
 846 for safety critical applications (such as self-driving cars) as they do not handle inten-
 847 tional or unintentional adversarial attacks. Moreover, as such adversarial examples
 848 exist in the physical world [106, 189], "the real world may be adversarial enough"
 849 [261] to fool such software.

850 2.2 Probabilistic and Nondeterministic Systems

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

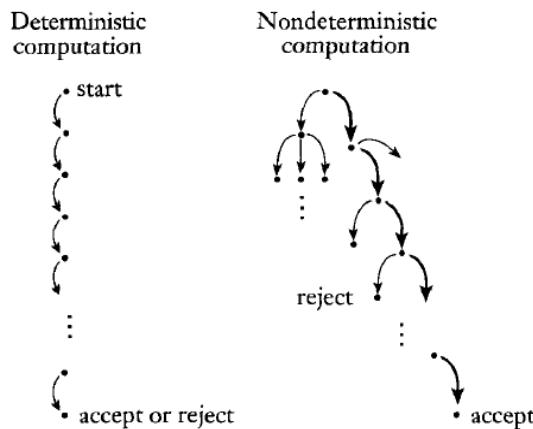


Figure 2.5: A deterministic system (left) always returns the same result in the same amount of steps. A nondeterministic system does not guarantee the same outcome, even with the same input data. Source: [110].

856 2.2.1 Interpreting the Uninterpretable

857 As the rise of applied AI increases, the need for engineering interpretability around
 858 models becomes paramount, chiefly from an external quality perspective that the
 859 *reliability* of the system can be inspected by end-users. Model interpretability has
 860 been stressed since early machine learning research in the late 1980s and 1990s (such
 861 as Quinlan [268] and Michie [226]), and although there has since been a significant
 862 body of work in the area [14, 29, 47, 60, 91, 108, 117, 127, 168, 198, 202, 212, 256,
 863 273, 286, 306, 331, 334], it is evident that ‘accuracy’ or model ‘confidence’ is still
 864 used as a primary criterion for AI evaluation [152, 162, 308]. Much research into
 865 neural network (NN) or support vector machine (SVM) development stresses that
 866 ‘good’ models are those with high accuracy. However, is accuracy enough to justify
 867 a model’s quality?

868 To answer this, we revisit what it means for a model to be accurate. Accuracy
 869 is an indicator for estimating how well a model’s algorithm will work with future
 870 or unforeseen data. It is quantified in the AI testing stage, whereby the algorithm
 871 is tested against cases known by humans to have ground truth but such cases are
 872 unknown by the algorithm. In production, however, all cases are unknown by both
 873 the algorithm *and* the humans behind it, and therefore a single value of quality is “not
 874 reliable if the future dataset has a probability distribution significantly different from
 875 past data” [113], a problem commonly referred to as the *datashift* problem [292].
 876 Analogously, Freitas [113] provides the following description of the problem:

877 *The military trained [a NN] to classify images of tanks into enemy
 878 and friendly tanks. However, when the [NN] was deployed in the field
 879 (corresponding to “future data”), it had a poor accuracy rate. Later,
 880 users noted that all photos of friendly (enemy) tanks were taken on a
 881 sunny (overcast) day. I.e., the [NN] learned to discriminate between
 882 the colors of the sky in sunny vs. overcast days! If the [NN] had
 883 output a comprehensible model (explaining that it was discriminating
 884 between colors at the top of the images), such a trivial mistake would
 885 immediately be noted.* [113]

886 So, why must we interpret models? While the formal definition of what it means
 887 to be *interpretable* is still somewhat disparate (though some suggestions have been
 888 proposed [202]), what is known is (i) there exists a critical trade-off between accuracy
 889 and interpretability [96, 112, 134, 167, 174, 355], and (ii) a single quantifiable value
 890 cannot satisfy the subjective needs of end-users [113]. As ever-growing domains
 891 ML become widespread², these applications engage end-users for real-world goals,
 892 unlike the aims in early ML research where the aim was to get AI working in the
 893 first place. In safety-critical systems where AI provide informativeness to humans
 894 to make the final call (see [65, 153, 176]), there is often a mismatch between the
 895 formal objectives of the model (e.g., to minimise error) and complex real-world
 896 goals, where other considerations (such as the human factors and cognitive science

²In areas such as medicine [28, 60, 104, 163, 168, 193, 257, 275, 331, 351, 358], bioinformatics [95, 114, 165, 173, 316], finance [14, 93, 153] and customer analytics [198, 334].

897 behind explanations³) are not realised: model optimisation is only worthwhile if they
898 “actually solve the original [human-centred] task of providing explanation” [236]
899 to end-users. **Therefore, when human-decision makers must be interpretable**
900 **themselves [276], any AI they depend on must also be interpretable.**

901 Recently, discussion behind such a notion to provide legal implications of in-
902 terpretability is topical. Doshi-Velez et al. [99] discuss when explanations are not
903 provided from a legal stance—for instance, those affected by algorithmic-based de-
904 cisions have a ‘right to explanation’ [209, 335] under the European Union’s GDPR⁴.
905 But, explanations are not the only way to ensure AI accountability: theoretical
906 guarantees (mathematical proofs) or statistical evidence can also serve as guarantees
907 [99], however, in terms of explanations, what form they take and how they are proven
908 correct are still open questions [202].

909 2.2.2 Explanation and Communication

910 From a software engineering perspective, explanations and interpretability are, by
911 definition, inherently communication issues: what lacks here is a consistent interface
912 between the AI system and the person using it. The ability to encode ‘common
913 sense reasoning’ [216] into programs today has been achieved, but *decoding* that
914 information is what still remains problematic. At a high level, Shannon and Weaver’s
915 theory of communication [299] applies, just as others have done with similar issues in
916 the software engineering realm [229, 346] (albeit to the domain of visual notations).
917 Humans map the world in higher-level concepts easily when compared to AI systems:
918 while we think of a tree first (not the photons of light or atoms that make up the
919 tree), an algorithm simply sees pixels, and not the concrete object [99] and the AI
920 interprets the tree inversely to humans. Therefore, the interpretation or explanation
921 is done inversely: humans do not explain the individual neurons fired to explain their
922 predictions, and therefore the algorithmic transparent explanations of AI algorithms
923 (“*which neurons were fired to make this AI think this tree is a tree?*”) do not work
924 here.

925 Therefore, to the user (as mapped using Shannon and Weaver’s theory), an AI
926 pipeline (the communication *channel*) begins with a real-world concept, y , that acts
927 as an *information source*. This information source is fed in as a *message*, x , (as pixels)
928 to an AI system (the *transmitter*). The transmitter encodes the pixels to a prediction,
929 \hat{y} , the *signal* of the message. This signal is decoded by the *receiver*, an explanation
930 system, $e_x(x, \hat{y})$, that tailors the prediction with the given input data to the intended
931 end user (the *destination*) as an explanation, \tilde{y} , another type of *message*. Therefore,
932 the user only sees the channel as an input/output pipeline of real-world objects, y ,
933 and explanations, \tilde{y} , tailored to *them*, without needing to see the inner-mechanics of
934 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.

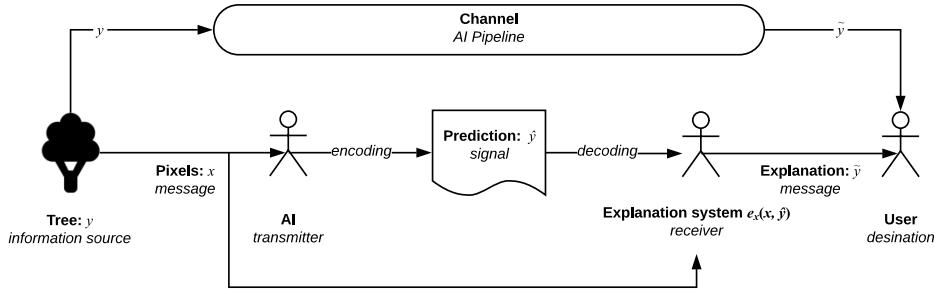


Figure 2.6: Theory of AI communication from information source, y , to intended user as explanations \tilde{y} .

935 2.2.3 Mechanics of Model Interpretation

936 How do we interpret models? Methods for developing interpretation models include:
 937 decision trees [53, 77, 140, 206, 269], decision tables [15, 198] and decision sets
 938 [191, 236]; input gradients, gradient vectors or sensitivity analysis [14, 195, 273,
 939 286, 297]; exemplars [115, 177]; generalised additive models [65]; classification
 940 (*if-then*) rules [49, 74, 250, 326, 348] and falling rule lists [306]; nearest neighbours
 941 [212, 270, 298, 344, 356] and Naïve Bayes analysis [28, 69, 107, 116, 144, 184, 193,
 942 358].

943 Cross-domain studies have assessed the interpretability of these techniques
 944 against end-users, measuring response time, accuracy in model response and user
 945 confidence [6, 114, 141, 153, 212, 291, 313, 334], although it is generally agreed
 946 that decision rules and decision tables provide the most interpretation in non-linear
 947 models such as SVMs or NNs [114, 212, 334]. For an extensive survey of the benefits
 948 and fallbacks of these techniques, we refer to Freitas [113], Doshi-Velez et al. [99]
 949 and Doshi-Velez and Kim [98].

950 2.3 Application Programming Interfaces

951 Application programming interfaces (APIs) are the interface between a developer
 952 needs and the software components at their disposal [10] by abstracting the underlying
 953 component behind a subroutine, protocol or specific tool. Therefore, it is natural
 954 to assess internal quality (and external quality if the software is in itself a service to
 955 be used by other developers—in this case an IWS) is therefore directly related to the
 956 quality the API offers [183].

957 Good APIs are known to be intuitive and require less documentation browsing
 958 [263], thereby increasing developer productivity. Conversely, poor APIs are those
 959 that are hard to interpret, thereby reducing developer productivity and product quality.
 960 The consequences of this have shown a higher demand of technical support (as
 961 measured in [145]) that, ultimately, causes the maintenance to be far more expensive,
 962 a phenomenon widely known in software engineering economics (see Section 2.1.1).

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

964 APIs for building desktop software, operating system APIs for interacting with the
965 operating system, remote APIs for communication of varying technologies through
966 common protocols, we focus on web APIs for communication of resources over
967 the web (being the common architecture of cloud-based services). Further infor-
968 mation on the development, usage and documentation of web APIs is provided in
969 Appendix A.1.

970 2.3.1 API Usability

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

976 Balancing the understanding of the information need (both conscious and un-
977 conscious), how to phrase that need and how to query it in an information retrieval
978 system is concept long studied in the information sciences [324]. In API design,
979 the most common form to convey knowledge to developers is through annotated
980 code examples and overviews to a platform's architectural and design decisions
981 [50, 97, 233, 280] though these studies have not effectively communicated *why* these
982 artefacts are important. What makes the developer *conceptually understand* these
983 artefacts?

984 Robillard and Deline [280] conducted a multi-phase, mixed-method approach to
985 create knowledge grounded in the professional experience of 440 software engineers
986 at Microsoft of varying experience to determine what makes APIs hard to learn,
987 the results of which previously published in an earlier report [279]. Their results
988 demonstrate that 'documentation-related obstacles' are the biggest hurdle in learning
989 new APIs. One of these implications are the *intent documentation* of an API (i.e.,
990 *what is the intent for using a particular API?*) and such documentation is required
991 only where correct API usage is not self-evident, where advanced uses of the API are
992 documented (but not the intent), and where performance aspects of the API impact
993 the application developed using it. They conclude that professional developers do
994 not struggle with learning the *mechanics* of the API, but in the *understanding* of how
995 the API fits in upwards to its problem domain and downward to its implementation:

996 *In the upwards direction, the study found that developers need help*
997 *mapping desired scenarios in the problem domain to the content of the*
998 *API, and in understanding what scenarios or usage patterns the API*
999 *provider intends and does not intend to support. In the downwards*
1000 *direction, developers want to understand how the API's implementation*
1001 *consumes resources, reports errors and has side effects. [280]*

1002 These results particularly corroborate to that of previous studies where devel-
1003 opers quote that they feel that existing learning content currently focuses on "how
1004 to do things, not necessarily *why*" [244]. This thereby reiterates the conceptual
1005 understanding of an API as paramount.

¹⁰⁰⁶ A later study by Ko and Riche [182] assessed the importance of a programmer's
¹⁰⁰⁷ conceptual understanding of the background behind the task before implementing the
¹⁰⁰⁸ task itself, a notion that we find most relevant for users of IWS APIs. While the study
¹⁰⁰⁹ did not focus on developing web APIs (rather implementing a Bluetooth application
¹⁰¹⁰ using platform-agnostic terminology), the study demonstrated how developers show
¹⁰¹¹ little confidence in their own metacognitive judgements to understand and assess the
¹⁰¹² feasibility of the intent of the API and understand the vocabulary and concepts within
¹⁰¹³ the domain (i.e., wireless connectivity). This indecision over what search results
¹⁰¹⁴ were relevant in their searches ultimately hindered their progress implementing the
¹⁰¹⁵ functionality, again decreasing productivity. Ko and Riche suggest to improve API
¹⁰¹⁶ usability by introducing the background of the API and its relevant concepts using
¹⁰¹⁷ glossaries linked to tutorials to each of the major concepts, and then relate it back to
¹⁰¹⁸ how to implement the particular functionality.

¹⁰¹⁹ Thus, an analysis of the conceptual understanding of IWS APIs by a range of
¹⁰²⁰ developers (from beginner to professional) is critical to best understand any differ-
¹⁰²¹ ences between existing studies and those that are nondeterministic. Our proposal is
¹⁰²² to perform similar survey research (see Chapter 3) in the search for further insight
¹⁰²³ into the developer's approach toward existing IWS APIs.

CHAPTER 3

1024

1025

1026

Research Methodology

1027

1028 < *todo: Revise this entire chapter for tense issues: JG - I did wonder about
1029 TENSE in Ch 3 - I didn't change but to think about - all this work
1030 is DONE so use either past (my pref) or present. Not "we propose
1031 to use..." etc etc all throughout. Especially for a by-papers thesis.
1032 Could revised to "we proposed to use ..." but I would suggest "We
1033 used ..." (my pref - past) or "We use..." (present). >*

1034 Investigating software engineering practices is often a complex task as it is im-
1035 perative to understand the social and cognitive processes around software engineers
1036 and not just the tools and processes used [103]. This chapter explores our research
1037 methodology by exploring five key elements of empirical software engineering re-
1038 search: firstly, (i) we provide an extended focus to the study by reviewing our research
1039 questions (see Section 1.4) anchored under the context of an existing research ques-
1040 tion classification taxonomy, (ii) characterise our research goals through an explicit
1041 philosophical stance, (iii) explain how the stance selected impacts our selection of
1042 research methods and data collection techniques (by dissecting our choice of meth-
1043 ods used to reach these research goals), (iv) discuss a set of criteria for assessing the
1044 validity of our study design and the findings of our research, and lastly (v) discuss
1045 the practical considerations of our chosen methods.

1046 The foundations for developing this research methodology has been expanded
1047 from that proposed by Easterbrook et al. [103], Wohlin and Aurum [349], Wohlin
1048 et al. [350] and Shaw [301].

1049 3.1 Research Questions Revisited

1050 To discuss our research strategy, we revisit our four primary and seven secondary
1051 research questions (RQs) through the classification technique discussed by Easter-
1052 brook et al. [103], a technique originally proposed in the field of psychology by

1053 Meltzoff and Cooper [221] but adapted to software engineering. A summary of the
1054 classifications made to our research questions are presented in Table 3.1.

1055 Our research study involves a mix of nine *empirical*¹ RQs, that focus on observing
1056 and analysing existing phenomena, and two *non-empirical* RQs, that focuses
1057 on designing better approaches to solve software engineering tasks [225]. The use
1058 of empirical *and* non-empirical RQs are best combined in long-term software en-
1059 gineering research studies where the phenomena are under-explored, as is the case
1060 with CVSs. Further, these approaches help propose solutions to issues found in the
1061 phenomena studied [347]. We discuss both our empirical and non-empirical RQs in
1062 Sections 3.1.1 and 3.1.2 below.

Table 3.1: A summary of our research questions classified using the strategies presented by Easterbrook et al. [103] and Meltzoff and Cooper [221].

#	RQ	Primary/ Secondary	RQ Classification
RQ1	What is the nature of cloud-based CVSs?	Primary	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 doc- ument, according to both literature and practi- tioners?	Secondary	EMPIRICAL ↪ Exploratory ↪ Composition
RQ2.2	What additional information or attributes do appli- cation 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 es- tablished 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 ro- bustness and reliability?	Primary	NON-EMPIRICAL ↪ Design

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

1063 3.1.1 Empirical Research Questions

1064 In total, we pose nine empirically-based RQs to help us understand the way develop-
1065 ers currently interact and work with web services that provide computer vision. The
1066 majority of these questions are *exploratory* questions that contribute to a landscape
1067 analysis of these services (RQ1, RQ1.1 and RQ1.2), how well they are documented
1068 (RQ2), and the issues developers currently face when using them (RQ3). Our other
1069 exploratory questions complement the answers to these questions. For instance, to
1070 understand if CVSs are sufficiently documented (an *existence* exploratory question
1071 posed in RQ2), we need to understand the components of a ‘sufficient’ or ‘com-
1072 plete’ API document via RQ2.1 as proposed in both the literature and by software
1073 developers. While RQ2.1 does not directly relate to CVSs, answering it gives us
1074 an understanding the components of complete API documentation, and therefore,
1075 we can assess what aspects they are missing and where improvements can be made
1076 (RQ2.2). These questions are *descriptive and classification* questions that help de-
1077 scribe and classify what practices are in use for existing CVS API documentation
1078 and the nature behind these services. Answering these exploratory questions assists
1079 in refining preciser terms of the phenomena, ways in which we find evidence for
1080 them, and ensuring the data found is valid.

1081 By answering these questions, we have a clearer understanding of the phenom-
1082 ena; we then follow up by posing two additional *base-rate questions* that helps
1083 provide a basis to confirm that the phenomena occurring is normal (or unusual)
1084 behaviour by investigating the patterns of phenomena’s occurrence against other
1085 phenomena. RQ3.1 is a *frequency and distribution* question to help us understand
1086 what types of issues developers often encounter most, given a lack of formal extended
1087 training in artificial intelligence. This achieves us an insight into the developer’s
1088 mindset and regular thought patterns toward these APIs. We can then contrast
1089 this distribution using our second base-rate question (RQ3.3), that assesses the
1090 distributional differences between these intelligent components and non-intelligent
1091 (conventional) software components. Combined, these two questions can help us
1092 answer how the issues raised against CVSs are different to normal Stack Overflow
1093 issues—our *descriptive-comparative* question posed in RQ3—and, similarly, we can
1094 classify and rank which issues developers find most frustrating (RQ3.2).

1095 3.1.2 Non-Empirical Research Questions

1096 RQ2.2 and RQ4 are both non-empirically-based *design questions*; they are con-
1097 cerned with ways in which we can improve a CVS by investigating what additional
1098 attributes are needed in both the documentation of CVSs and in the integration
1099 architectures developers can employ to improve reliability and robustness in their
1100 applications. They are not classified as empirical questions as we investigate what
1101 *will be* and not *what is*. By understanding the process by which developers desire
1102 additional attributes of documentation and integration strategies, we can help shape
1103 improvements to the existing designs of using CVSs.

3.2 Philosophical Stances

*1104 ⟨ todo: JG: do you really need this section? :-) ⟩ ⟨ todo: AC: I am not sure – I
1105 thought it would be good to anchor the research per advice from Raj ⟩*

*1106 Philosophical stances guide the researcher’s action by fortifying what constitutes
1107 ‘valid truth’ against a fundamental set of core beliefs [278]. In software engineer-
1108 ing, four dominant philosophical stances are commonly characterised [78, 259]:
1109 positivism (or post-positivism), constructivism (or interpretivism), pragmatism, and
1110 critical theory (or advocacy/participatory). To construct such a ‘validity of truth’,
1111 we will review these four philosophical stances in this section, and state the stance
1112 that we explicitly adopt and our reasoning for this.*

1113 3.2.0.1 Positivism

*1114 Positivists claim truth to be all observable facts, reduced piece-by-piece to smaller
1115 components which is incrementally verifiable to form truth. We do not base our
1116 work on the positivistic stance as the theories governing verifiable hypothesis must
1117 be precise from the start of the research. Moreover, due to its reductionist approach,
1118 it is difficult to isolate these hypotheses and study them in isolation from context.
1119 As our hypotheses are not context-agnostic, we steer clear from this stance.*

1120 3.2.0.2 Constructivism

*1121 Constructivists see knowledge embedded within the human context; truth is the
1122 *interpretive* observation by understanding the differences in human thought between
1123 meaning and action [181]. That is, the interpretation of the theory is just as important
1124 to the empirical observation itself. We partially adopt a constructivist stance as we
1125 attempt to model the developer’s mindset, being an approach that is rich in qualitative
1126 data on human activity.*

1127 3.2.0.3 Pragmatism

*1128 Pragmatism is a less dogmatic approach that encourages the incomplete and approx-
1129 imate nature of knowledge and is dependent on the methods in which the knowledge
1130 was extracted. The utility of consensually agreed knowledge is the key outcome, and
1131 is therefore relative to those who seek utility in the knowledge—what is the useful
1132 for one person is not so for the other. While we value the utility of knowledge, it is
1133 difficult to obtain consensus especially on an ill-researched topic such as ours, and
1134 therefore we do not adopt this stance.*

1135 3.2.0.4 Critical Theory

*1136 This study chiefly adopts the philosophy of critical theory [8]. A key outcome of
1137 the study is to shift the developer’s restrictive deterministic mindset and shed light
1138 on developing a new framework actively with the developer community that seeks
1139 to improve the process of using such APIs. In software engineering, critical theory
1140 is used to “actively [seek] to challenge existing perceptions about software practice”*

[103], and this study utilises such an approach to shift the mindset of CVS consumers and providers alike on how the documentation and metadata should not be written with the ‘traditional’ deterministic mindset at heart. Thus, our key philosophical approach is critical theory to seek out *what-can-be* using partial constructivism to model the current *what-is*.

3.3 Research Methods

Research methods are “a set of organising principles around which empirical data is collection and analysed” [103]. Creswell [78] suggests that strong research design is reflected when the weaknesses of multiple methods complement each other. Using a mixed-methods approach is therefore commonplace in software engineering research, typically due to the human-oriented nature investigating how software engineers work both individually (where methods from psychology may be employed) and together (where methods from sociology may be employed).

Therefore, studies in software engineering are typically performed as field studies where researchers and developers (or the artefacts they produce) are analysed either directly or indirectly [305]. The mixed-methods approach combines five classes of field study methods (or empirical strategies/studies) most relevant in empirical software engineering research [103, 172, 350]: controlled experiments, case studies, survey research, ethnographies, and action research. We chiefly adopt a mixed-methods approach to our work using the *concurrent triangulation* mixed-methods strategy [214] as it best compensates for weaknesses that exist in all research methods, and employs the best strengths of others [78].

3.3.1 Review of Relevant Research Methods

Below we review some of the research methods most relevant to our research questions as refined in Section 3.1 as presented by Easterbrook et al. [103].

3.3.1.1 Controlled Experiments

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

While we have well-defined RQs, refining them into precise, *measurable* variables is challenging due to the qualitative nature they present. A well-defined population is also critical and must be easily accessible; the varied range of beginner to expert software engineers with varied understanding of artificial intelligence concepts is required to perform controlled experiments, and thus recruitment may

1182 prove challenging. Lastly, the controlled experiment is essentially reductionist by
1183 affecting a small amount of variables of interest and controlling all others. This
1184 approach is too clinical for the practical outcomes by which our research goals aim
1185 for, and is therefore closely tied to the positivist stance.

1186 *3.3.1.2 Case Studies*

1187 Case studies investigate phenomena in their real-life context and are well-suited
1188 when the boundary between context and phenomena is unknown [354]. They offer
1189 understanding of how and why certain phenomena occur, thereby investigating ways
1190 cause-effect relationships can occur. They can be used to test existing theories
1191 (*confirmatory case studies*) by refuting theories in real-world contexts instead of
1192 under laboratory conditions or to generate new hypotheses and build theories during
1193 the initial investigation of some phenomena (*exploratory case studies*).

1194 Case studies are well-suited where the context of a situation plays a role in
1195 the phenomenon being studied. They also lend themselves to purposive sampling
1196 rather than random sampling, and thus it is possible to selectively choose cases that
1197 benefit our research goals and (using our critical theorist stance) select cases that
1198 will actively benefit our participant software engineering audience most to draw
1199 attention to situations regarded as problematic in CVS.

1200 *3.3.1.3 Survey Research*

1201 Survey research identifies characteristics of a broad population of individuals through
1202 direct data collection techniques such as interviews and questionnaires or indepen-
1203 dent techniques such as data logging. Defining that well-defined population is
1204 critical, and selecting a representative sample from it to generalise the data gathered
1205 usually assists in answering base-rate questions.

1206 By identifying representative sample of the population, from beginner to ex-
1207 perienced developers with varying understanding of CVS APIs, we can use survey
1208 research to assist in answering our exploratory and base-rate RQs (see Section 3.1.1)
1209 in determining the qualitative aspects of how individual developers perceive and
1210 work with the existing APIs, either by directly asking them, or by mining third-party
1211 discussion websites such as Stack Overflow (SO). Similarly, we can use this strategy
1212 to assess the developer’s understanding on what makes API documentation sufficient
1213 by assessing whether specific factors suggested from literature are useful according
1214 to developers. However, with direct survey research techniques, low response rates
1215 may prove challenging, especially if no inducements can be offered for participation.

1216 *3.3.1.4 Ethnographies*

1217 Ethnographies investigates the understanding of social interaction within community
1218 through field observation [282]. Resulting ethnographies help understand how soft-
1219 ware engineering technical communities build practices, communication strategies
1220 and perform technical work collaboratively.

1221 Ethnographies require the researcher to be highly trained in observational and
1222 qualitative data analysis, especially if the form of ethnography is participant observa-
1223 tion, whereby the researcher is embedded of the technical community for observation.
1224 This may require the longevity of the study to be far greater than a couple of weeks,
1225 and the researcher must remain part of the project for its duration to develop enough
1226 local theories about how the community functions. While it assists in revealing
1227 subtle but important aspects of work practices within software teams, this study
1228 does not focus on the study of teams, and is therefore not a research method relevant
1229 to this project.

1230 **3.3.1.5 Action Research**

1231 Action researchers simultaneously solve real-world problems while studying the
1232 experience of solving the problem [89] by actively seeking to intervene in the
1233 situation for the purpose of improving it. A precondition is to engage with a
1234 *problem owner* who is willing to collaborate in identifying and solving the problem
1235 faced. The problem must be authentic (a problem worth solving) and must have
1236 new knowledge outcomes for those involved. It is also characterised as an iterative
1237 approach to problem solving, where the knowledge gained from solving the problem
1238 has a desirable solution that empowers the problem owner and researcher.

1239 This research is most associated to our adopted philosophical stance of critical
1240 theory. As this project is being conducted under the Applied Artificial Intelligence
1241 Institute (A^2I^2) collaboratively with engaged industry clients, we have identified a
1242 need for solving an authentic problem that industry faces. The desired outcome
1243 of this project is to facilitate wider change in the usage and development of CVSS;
1244 thus, engaging action research as a potential method throughout the mixed-methods
1245 approach is used in this research.

1246 **3.3.2 Review of Data Collection Techniques for Field Studies**

1247 Singer et al. developed a taxonomy [196, 305] showcasing data collection techniques
1248 in field studies that are used in conjunction with a variety of methods based on the
1249 level of interaction between researcher and software engineer, if any. This taxonomy
1250 is reproduced in Figure 3.1.

1251 **3.4 Research Design**

1252 This section discusses an overview of the design of methods used within the experi-
1253 ments conducted under this thesis. For each experiment, we describe an overview of
1254 the experiment grounded known methods and techniques (Sections 3.3.1 and 3.3.2)
1255 and our approach to analysing the data, as well as relating the selecting method back
1256 to a specific RQ. Details of each experiment presented in this thesis, the coherency
1257 between them, and where they can be found are given in Sections 1.6 and 1.7.

Figure 3.1: Questions asked by software engineering researchers (column 2) that can be answered by field study techniques. (From [305].)

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.

1258 3.4.1 Landscape Analysis of Computer Vision Services

1259 To understand the behavioural and evolutionary profiles of CVSs (i.e., RQ1), we em-
1260 ployed a longitudinal study based around a dynamic system analysis [305]. Specif-
1261 ically, we used structured observations of three services using the same dataset to
1262 understand how the responses from these services change with time. Lastly, we
1263 utilised documentation analysis to assess the overall ‘picture’ of how these ser-
1264 vices are documented. Further details on this experiment is given in **Chapter 4,**
1265 **Section 4.4.**

1266 3.4.2 Utility of API Documentation in Computer Vision Services

1267 To assess whether these services are sufficiently documented (i.e., RQ2), we con-
1268 ducted a systematic mapping study [178, 260] of the various academic sources
1269 detailing API documentation knowledge. We then consolidated this information
1270 into a structured taxonomy following a systematic taxonomy development method
1271 specific to software engineering studies [330].

1272 We then followed the triangulation approach proposed by Mayring [214] to val-
1273 idate the taxonomy by use of a personal opinion survey. Kitchenham and Pfleeger
1274 [179] provide an introduction on methods used to conduct personal opinion surveys
1275 which we adopted as an initial reference in (i) shaping our survey objectives around
1276 our research goals, (ii) designing a cross-sectional survey, (iii) developing and eval-
1277 uating our survey instrument, (iv) evaluating our instruments, (v) obtaining the data
1278 and (vi) analysing the data. We adapted Brooke’s systematic usability scale [55]
1279 technique by basing our research questions against a known surveying instrument.

1280 As is good practice in developing questionnaire instruments to evaluate their re-
1281 liability and validity [203], we evaluated our instrument design by asking colleagues
1282 to critique it via pilot studies within A²I². This assisted in identifying any problems
1283 with the questionnaire itself and with any issues that may have occurred with the
1284 response rate and follow-up procedures.

1285 Findings from the pilot study helped inform us for a widely distributed question-
1286 naire using snow-balling sampling. Ethics approval from the Faculty of Science,
1287 Engineering and Built Environment Human Ethics Advisory Group (SEBE HEAG)
1288 was approved to externally conduct this survey research (see Appendix E). Further
1289 details on these methods are detailed within **Chapter 7, Section 7.3.**

1290 3.4.3 Developer Issues concerning Computer Vision Services

1291 Developers typically congregate in search of discourses on issues they face in online
1292 forums, such as Stack Overflow (SO) and Quora, as well as writing their experiences
1293 in personal blogs such as Medium. The simplest of these platforms is SO (a sub-
1294 community of the Stack Exchange family of targeted communities) that specifically
1295 targets developer issues on using a simple Q&A interface, where developers can
1296 discuss technical aspects and general software development topics. Moreover, SO
1297 is often acknowledged as *the* ‘go-to’ place for developers to find high-quality code
1298 snippets that assist in their problems [314].

1299 Thus, to begin understanding the issues developers face when using CVSs and
1300 whether there is a substantial difference to conventional domains (i.e., RQ3), we
1301 used repository mining on SO to help answer RQ3. Specifically, we selected SO
1302 due to its targeted community of developers² and the availability of its publicly
1303 available dataset released as ‘data dumps’ on the Stack Exchange Data Explorer³
1304 and Google BigQuery⁴. Studies conducted have also used SO to mine developer
1305 discourse [7, 17, 23, 71, 201, 241, 251, 271, 283, 307, 320, 338]. Further details on
1306 how we approached the design for this study can be found in **Chapter 5, Section 5.4**
1307 and **Chapter 6, Section 6.3**.

1308 3.4.4 Designing Improved Integration Strategies

1309 Our improved integration strategies (i.e., RQ4) evolved organically over the dura-
1310 tion of this research through the use of industry case studies and action research.
1311 We developed several iterative prototypes to the integration strategies and used a
1312 mix of statistical and technical evaluations to analyse whether our improved in-
1313 tegration strategies can prove useful. Further details about these approaches are
1314 detailed in **Chapter 8, Section 8.5.1** and **Chapter 9, Section 9.3** and **Chapter 10,**
1315 **Section 10.5.**

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

1316

Part II

1317

Publications

CHAPTER 4

1318

1319

1320

Identifying Evolution in Computer Vision Services[†]

1321

1322 **Abstract** Recent advances in artificial intelligence (AI) and machine learning (ML), such
1323 as computer vision, are now available as intelligent web services (IWSs) and their acces-
1324 sibility and simplicity is compelling. Multiple vendors now offer this technology as cloud
1325 services and developers want to leverage these advances to provide value to end-users. How-
1326 ever, there is no firm investigation into the maintenance and evolution risks arising from use
1327 of these IWSs; in particular, their behavioural consistency and transparency of their function-
1328 ality. We evaluated the responses of three different IWSs (specifically computer vision) over
1329 11 months using 3 different data sets, verifying responses against the respective documenta-
1330 tion and assessing evolution risk. We found that there are: (1) inconsistencies in how these
1331 services behave; (2) evolution risk in the responses; and (3) a lack of clear communication
1332 that documents these risks and inconsistencies. We propose a set of recommendations to
1333 both developers and IWS providers to inform risk and assist maintainability.

4.1 Introduction

1334 The availability of intelligent web services (IWSs) has made artificial intelligence
1335 (AI) tooling accessible to software developers and promises a lower entry barrier for
1336 their utilisation. Consider state-of-the-art computer vision analysers, which require
1337 either manually training a deep-learning classifier, or selecting a pre-trained model
1338 and deploying these into an appropriate infrastructure. Either are laborious in time,
1339 and require non-trivial expertise along with a large data set when training or customisa-
1340 tion is needed. In contrast, IWSs providing computer vision (i.e., computer vision
1341 services or CVSs such as [363, 375, 376, 377, 384, 388, 396, 397, 398, 402, 415,

[†]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.

416, 449, 450]) abstract these complexities behind a web application programming
1343 interface (API) call. This removes the need to understand the complexities required
1344 of machine learning (ML), and requires little more than the knowledge on how to
1345 use RESTful endpoints. The ubiquity of these services is exemplified through their
1346 rapid uptake in applications such as aiding the vision-impaired [88, 272].
1347

1348 While IWSs have seen quick adoption in industry, there has been little work
1349 that has considered the software quality perspective of the risks and impacts posed
1350 by using such services. In relation to this, there are three main challenges: (1)
1351 incorporating stochastic algorithms into software that has traditionally been deter-
1352 ministic; (2) the general lack of transparency associated with the ML models; and
1353 (3) communicating to application developers.

1354 ML typically involves use of statistical techniques that yield components with
1355 a non-deterministic external behaviour; that is, for the same given input, different
1356 outcomes may result. However, developers, in general, are used to libraries and small
1357 components behaving predictably, while systems that rely on ML techniques work
1358 on confidence intervals¹ and probabilities. For example, the developer’s mindset
1359 suggests that an image of a border collie—if sent to three intelligent computer vision
1360 services (CVSs)—would return the label ‘dog’ consistently with time regardless
1361 of which service is used. However, one service may yield the specific dog breed,
1362 ‘border collie’, another service may yield a permutation of that breed, ‘collie’, and
1363 another may yield broader results, such as ‘animal’; each with results of varying
1364 confidence values.² Furthermore, the third service may evolve with time, and
1365 thus learn that the ‘animal’ is actually a ‘dog’ or even a ‘collie’. The outcomes
1366 are thus behaviourally inconsistent between services providing conceptually similar
1367 functionality. As a thought exercise, consider if the sub-string function were created
1368 using ML techniques—it would perform its operation with a confidence where the
1369 expected outcome and the AI inferred output match as a *probability*, rather than a
1370 deterministic (constant) outcome. How would this affect the developers’ approach
1371 to using such a function? Would they actively take into consideration the non-
1372 deterministic nature of the result?

1373 Myriad software quality models and software engineering practices advocate
1374 maintainability and reliability as primary characteristics; stability, testability, fault
1375 tolerance, changeability and maturity are all concerns for quality in software com-
1376 ponents [148, 266, 309] and one must factor these in with consideration to soft-
1377 ware evolution challenges [130, 131, 223, 224, 325]. However, the effect this
1378 non-deterministic behaviour has on quality when masked behind an IWS is still
1379 under-explored to date in software engineering literature, to our knowledge. Where
1380 software depends on IWSs to achieve functionality, these quality characteristics may
1381 not be achieved, and developers need to be wary of the unintended side effects and
1382 inconsistency that exists when using non-deterministic components. A CVS may
1383 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.

1384 sis, but developers are more likely to approach IWSs with a mindset that anticipates
1385 consistency. Although the documentation does hint at this non-deterministic be-
1386 haviour (i.e., the descriptions of ‘confidence’ in various CVSs suggest the they are
1387 not always confident, and thus not deterministic [361, 386, 403]), the integration
1388 mechanisms offered by popular vendors do not seem to fully expose the nuances,
1389 and developers are not yet familiar with the trade-offs.

1390 Do popular CVSs, as they currently stand, offer consistent behaviour, and if not,
1391 how is this conveyed to developers (if it is at all)? If CVSs are to be used in production
1392 services, do they ensure quality under rigorous service quality assurance (SQA)
1393 frameworks [148]? What evolution risk [130, 131, 223, 224] do they pose if these
1394 services change? To our knowledge, few studies have been conducted to investigate
1395 these claims. This paper assesses the consistency, evolution risk and consequent
1396 maintenance issues that may arise when developers use IWSs. We introduce a
1397 motivating example in Section 4.2, discussing related work and our methodology
1398 in Sections 4.3 and 4.4. We present and interpret our findings in Section 4.5. We
1399 argue with quantified evidence that these IWSs can only be considered with a mature
1400 appreciation of risks, and we make a set of recommendations in Section 4.6.

1401 4.2 Motivating Example

1402 Consider Rosa, a software developer, who wants to develop a social media photo-
1403 sharing mobile app that analyses her and her friends photos on Android and iOS.
1404 Rosa wants the app to categorise photos into scenes (e.g., day vs. night, outdoors
1405 vs. indoors), generate brief descriptions of each photo, and catalogue photos of her
1406 friends as well as common objects (e.g., all photos with a dog, all photos on the
1407 beach).

1408 Rather than building a computer vision engine from scratch, Rosa thinks she
1409 can achieve this using one of the popular CVSs (e.g., [363, 375, 376, 377, 384, 388,
1410 396, 397, 398, 402, 415, 416, 449, 450]). However, Rosa comes from a typical
1411 software engineering background with limited knowledge of the underlying deep-
1412 learning techniques and implementations as currently used in computer vision. Not
1413 unexpectedly, she internalises a mindset of how such services work and behave based
1414 on her experience of using software libraries offered by various SDKs. This mindset
1415 assumes that different cloud vendor image processing APIs more-or-less provide
1416 similar functionality, with only minor variations. For example, cloud object storage
1417 for Amazon S3 is both conceptually and behaviourally very similar to that of Google
1418 Cloud Storage or Azure Storage. Rosa assumes the CVSs of these platforms will,
1419 therefore, likely be very similar. Similarly, consider the string libraries Rosa will
1420 use for the app. The conceptual and behavioural similarities are consistent; a string
1421 library in Java (Android) is conceptually very similar to the string library she will
1422 use in Swift (iOS), and likewise both behave similarly by providing the same results
1423 for their respective sub-string functionality. However, **unlike the cloud storage and**
1424 **string libraries, different CVSs often present conceptually similar functionality**
1425 **but are behaviourally very different.** IWS vendors also hide the depth of knowledge
1426 needed to use these effectively—for instance, the training data set and ontologies

1427 used to create these services are hidden in the documentation. Thus, Rosa isn't even
1428 exposed to this knowledge as she reads through the documentation of the providers
1429 and, thus, Rosa makes the following assumptions:

- 1430 • **"I think the responses will be consistent amongst these CVSSs."** When Rosa
1431 uploads a photo of a dog, she would expect them all to respond with 'dog'. If
1432 Rosa decides to switch which service she is using, she expects the ontologies
1433 to be compatible (all CVSSs *surely* return dog for the same image) and therefore
1434 she can expect to plug-in a different service should she feel like it making only
1435 minor code modifications such as which endpoints she is relying on.
- 1436 • **"I think the responses will be constant with time."** When Rosa uploads the
1437 photo of a dog for testing, she expects the response to be the same in 10 weeks
1438 time once her app is in production. Hence, in 10 weeks, the same photo of the
1439 dog should return the same label.

1440 4.3 Related Work

1441 If we were to view CVSSs through the lenses of an SQA framework, robustness,
1442 consistency, and maintainability often feature as quality attributes in myriad soft-
1443 ware quality models (e.g., [159]). Software quality is determined from two key
1444 dimensions: (1) in the evaluation of the end-product (external quality) and (2) the
1445 assurances in the development processes (internal quality) [266]. We discuss both
1446 perspectives of quality within the context of our work in this section.

1447 4.3.1 External Quality

1448 4.3.1.1 Robustness for safety-critical applications

1449 A typical focus of recent work has been to investigate the robustness of deep-
1450 learning within computer vision technique implementation, thereby informing the
1451 effectiveness in the context of the end-product. The common method for this has
1452 been via the use of adversarial examples [317], where input images are slightly
1453 perturbed to maximise prediction error but are still interpretable to humans.

1454 Google Cloud Vision, for instance, fails to correctly classify adversarial examples
1455 when noise is added to the original images [149]. Rosenfeld et al. [285] illustrated
1456 that inserting synthetic foreign objects to input images (e.g., a cartoon elephant)
1457 can completely alter classification output. Wang et al. [336] performed similar
1458 attacks on a transfer-learning approach of facial recognition by modifying pixels of
1459 a celebrity's face to be recognised as a completely different celebrity, all while still
1460 retaining the same human-interpretable original celebrity. Su et al. [312] used the
1461 ImageNet database to show that 41.22% of images drop in confidence when just a
1462 *single pixel* is changed in the input image; and similarly, Eykholt et al. [106] recently
1463 showed similar results that made a convolutional neural network (CNN) interpret a
1464 stop road-sign (with mimicked graffiti) as a 45mph speed limit sign.

1465 The results suggest that current state-of-the-art computer vision techniques may
1466 not be robust enough for safety critical applications as they do not handle intentional

1467 or unintentional adversarial attacks. Moreover, as such adversarial examples exist in
1468 the physical world [106, 189], “the natural world may be adversarial enough” [261]
1469 to fool AI software. Though some limitations and guidelines have been explored
1470 in this area, the perspective of *Intelligent Web Services* is yet to be considered and
1471 specific guidelines do not yet exist when using CVSS.

1472 **4.3.1.2 Testing strategies in ML applications**

1473 Although much work applies ML techniques to automate testing strategies, there is
1474 only a growing emphasis that considers this in the opposite sense; that is, testing
1475 to ensure the ML product works correctly. There are few reliable test oracles
1476 that ensure if an ML has been implemented to serve its algorithm and use case
1477 purposefully; indeed, “the non-deterministic nature of many training algorithms
1478 makes testing of models even more challenging” [11]. Murphy et al. [232] proposed
1479 a software engineering-based testing approach on ML ranking algorithms to evaluate
1480 the ‘correctness’ of the implementation on a real-world data set and problem domain,
1481 whereby discrepancies were found from the formal mathematical proofs of the ML
1482 algorithm and the implementation.

1483 Recently, Braiek and Khomh [48] conducted a comprehensive review of testing
1484 strategies in ML software, proposing several research directions and recommendations
1485 in how best to apply software engineering testing practices in ML programs.
1486 However, much of the area of this work specifically targets ML engineers, and not
1487 application developers. Little has been investigated on how application developers
1488 perceive and understand ML concepts, given a lack of formal training; we note that
1489 other testing strategies and frameworks proposed (e.g., [52, 231, 240]) are targeted
1490 chiefly to the ML engineer, and not the application developer.

1491 However, Arpteg et al. [11] recently demonstrated (using real-world ML projects)
1492 the developmental challenges posed to developers, particularly those that arise when
1493 there is a lack of transparency on the models used and how to troubleshoot ML
1494 frameworks using traditional software engineering debugging tools. This said, there
1495 is no further investigations into challenges when using the higher, ‘ML friendly’
1496 layers (e.g., IWSs) of the ‘machine learning spectrum’ [248], rather than the ‘lower
1497 layers’ consisting of existing ML frameworks and algorithms targeted toward the
1498 ML community.

1499 **4.3.2 Internal Quality**

1500 **4.3.2.1 Quality metrics for cloud services**

1501 CVSS are based on cloud computing fundamentals under a subset of the Platform as
1502 a Service (PaaS) model. There has been work in the evaluation of PaaS in terms of
1503 quality attributes [120]: these attributes are exposed using service-level agreements
1504 (SLAs) between vendors and customers, and customers denote their demanded
1505 quality of service (QoS) to ensure the cloud services adhere to measurable KPI
1506 attributes.

1507 Although, popular services, such as cloud object storage, come with strong QoS

1508 agreement, to date IWSs do not come with deep assurances around their performance
1509 and responses, but do offer uptime guarantees. For example, how can Rosa demand
1510 a QoS that ensures all photos of dogs uploaded to her app guarantee the specific dog
1511 breeds are returned so that users can look up their other friend's 'border collie's?
1512 If dog breeds are returned, what ontologies exist for breeds? Are they consistent
1513 with each other, or shortened? ('Collie' versus 'border collie'; 'staffy' versus
1514 'staffordshire bull terrier')? For some applications, these unstated QoS metrics
1515 specific to the ML service may have significant legal ramifications.

1516 4.3.2.2 *Web service documentation and documenting ML*

1517 From the *developer's* perspective, little has been achieved to assess IWS quality
1518 or assure quality of these CVSs. Web services and their APIs are the bridge be-
1519 tween developers' needs and the software components [10]; therefore, assessing
1520 such CVSs from the quality of their APIs is thereby directly related to the develop-
1521 ment quality [183]. Good APIs should be intuitive and require less documentation
1522 browsing [263], thereby increasing productivity. Conversely, poor APIs that are
1523 hard to understand and work with reduce developer productivity, thereby reducing
1524 product quality. This typically leads to developers congregating on forums such as
1525 Stack Overflow, leading to a repository of unstructured knowledge likely to concern
1526 API design [340]. The consequences of addressing these concerns in development
1527 leads to a higher demand in technical support (as measured in [145]) that, ultimately,
1528 causes the maintenance to be far more expensive, a phenomenon widely known in
1529 software engineering economics [42]. Rosa, for instance, isn't aware of technical ML
1530 concepts; if she cannot reason about what search results are relevant when brows-
1531 ing the service and understanding functionality, her productivity is significantly
1532 decreased. Conceptual understanding is critical for using APIs, as demonstrated by
1533 Ko and Riche, and the effects of maintenance this may have in the future of her
1534 application is unknown.

1535 Recent attempts to document attributes and characteristics on ML models have
1536 been proposed. Model cards were introduced by Mitchell et al. [228] to describe how
1537 particular models were trained and benchmarked, thereby assisting users to reason
1538 if the model is right for their purposes and if it can achieve its stated outcomes.
1539 Gebru et al. [124] also proposed datasheets, a standardised documentation format to
1540 describe the need for a particular data set, the information contained within it and
1541 what scenarios it should be used for, including legal or ethical concerns.

1542 However, while target audiences for these documents may be of a more technical
1543 AI level (i.e., the ML engineer), there is still no standardised communication format
1544 for application developers to reason about using particular IWSs, and the ramifica-
1545 tions this may have on the applications they write is not fully conveyed. Hence, our
1546 work is focused on the application developer perspective.

4.4 Method

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

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

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

[RH3] CVSs do not effectively communicate this evolution and instability, introducing risk into engineering these systems.

We conducted two experiments to address these hypotheses against three popular CVSs: AWS Rekognition [363], Google Cloud Vision [388], Azure Computer Vision [402]. Specifically, we targeted the AWS DetectLabels endpoint [361], the Google Cloud Vision annotate:images endpoint [386] and Azure's analyze endpoint [403]. For the remainder of this paper, we de-identify our selected CVSs by labelling them as services A, B and C but do not reveal mapping to prevent any implicit bias. Our selection criteria for using these particular three services are based on the weight behind each service provider given their prominence in the industry (Amazon, Google and Microsoft), the ubiquity of their hosting cloud platforms as industry leaders of cloud computing (i.e., AWS, Google Cloud and Azure), being in the top three most adopted cloud vendors in enterprise applications in 2018 [277] and the consistent popularity of discussion amongst developers in developer communities such as Stack Overflow. While we choose these particular cloud CVSs, we acknowledge that similar services [376, 377, 384, 397, 398, 449, 450] also exist, including other popular services used in Asia [375, 396, 415, 416] (some offering 3D image analysis [374]). We reflect on the impacts this has to our study design in Section 4.7.

Our study involved an 11-month longitudinal study which consisted of two 13 week and 17 week experiments from April to August 2018 and November 2018 to March 2019, respectively. Our investigation into documentation occurred on August 28 2018. In total, we assessed the services with three data sets; we first ran a pilot study using a smaller pool of 30 images to confirm the end-points remain stable, re-running the study with a larger pool of images of 1,650 and 5,000 images. Our selection criteria for these three data sets were that the images had to have varying objects, taken in various scenes and various times. Images also needed to contain disparate objects. Our small data set was sourced by the first author by taking photos of random scenes in an afternoon, whilst our second data set was sourced from 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 [200]. We have made our other two data sets available online ([379]). We collected results and their responses from each service’s API endpoint using a python script [383] 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

1626 (6) **Consistency of the top label’s confidence over time.** The top label of an
 1627 image may remain static from one week to the next for the same service, but
 1628 its confidence values may change with time. Therefore, there is inconsistency
 1629 in the top label’s confidence due to model evolution.

1630 For the above aspects of consistency, we calculated the spread of variation for the
 1631 top label’s confidences of each service for every 1 percent point; that is, the frequency
 1632 of top label confidences within 100–99%, 99–98% etc. The consistency of top label’s
 1633 and their confidences between each service was determined by intersecting the labels
 1634 of each service per image and grouping the intersecting label’s confidences together.
 1635 This allowed us to determine relevant probability distributions. For reproducibility,
 1636 all quantitative analysis is available online [380].

1637 4.5 Findings

1638 4.5.1 Consistency of top labels

1639 4.5.1.1 Consistency across services

1640 Table 4.2 presents the consistency of the top labels between data sets, as measured
 1641 by the cardinality of the intersection of all three services’ set of top labels divided
 1642 by the number of images per data set. A combination of services present varied
 1643 overlaps in their top labels; services A and C provide the best overlap for all three
 1644 data sets, however the intersection of all three irrespective of data sets is low.

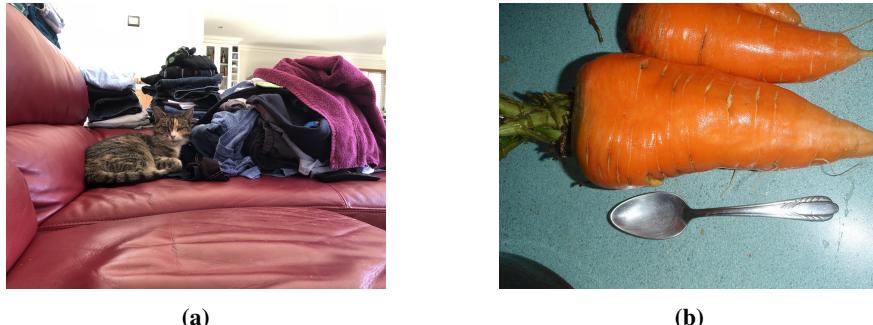


Figure 4.2: *Left:* The top labels for each service do not intersect, with each having a varied ontology: $T_i = \{ A = \{ \text{'black'} \}, B = \{ \text{'indoor'} \}, C = \{ \text{'slide'}, \text{'toy'} \} \}$. (Service C returns both ‘slide’ and ‘toy’ with equal confidence.) *Right:* The top labels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

1645 The implication here is that, without semantic comparison (see Section 4.7),
 1646 service vendors are not ‘plug-and-play’. If Rosa uploaded the sample images in
 1647 this paper to her application to all services, she would find that only Figure 4.1
 1648 responds with ‘person’ for services B and C in their respective set of top labels.
 1649 However, if she decides to then adopt service A, then Figure 4.1’s top label becomes
 1650 ‘conversation’; the ‘person’ label does not appear within the top 15 labels for service
 1651 A and, conversely, the ‘conversation’ label does not appear in the other services top
 1652 15.

1653 Should she decide if the performance of a particular service isn’t to her needs,
 1654 then the vocabulary used for these labels becomes inconsistent for all other images;
 1655 that is, the top label sets per service for Figure 4.2a shows no intersection at all.
 1656 Furthermore, the part of the image each service focuses on may not be consistent
 1657 for their top labels; in Figure 4.2b, service A’s top label focuses on the vegetable
 1658 (‘carrot’), service C focuses on the ‘spoon’, while service B’s focus is that the image
 1659 is ‘indoor’s. It is interesting to note that service B focuses on the scene matter
 1660 (indoors) rather than the subject matter. (Furthermore, we do not actually know if
 1661 the image in Figure 4.2b was taken indoors.)

1662 Hence, developers should ensure that the vocabulary used by a particular service
 1663 is right for them before implementation. As each service does not work to the
 1664 same standardised model, trained with disparate training data, and tuned differently,
 1665 results will differ despite the same input. This is unlike deterministic systems: for
 1666 example, switching from AWS Object Storage to Google Cloud Object storage will
 1667 conceptually provide the same output (storing files) for the same input (uploading
 1668 files). However, CVSs do not agree on the top label for images, and therefore
 1669 developers are likely to be vendor locked, making changes between services non-
 1670 trivial.

1671 4.5.1.2 Semantic consistency where $|T_i| > 1$

1672 Service C returns two top labels for Figure 4.2a; ‘slide’ and ‘toy’. More than one
 1673 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.

1674 images for all three data sets, respectively) though this also occurs in B in the large
 1675 (4.97% of all images) and COCOVal17 data sets (2.38%). Semantic inconsistencies
 1676 of what this label conceptually represents becomes a concern as these labels have
 1677 confidences of *equal highest* consistency. Thus, some services are inconsistent in
 1678 themselves and cannot give a guaranteed answer of what exists in an image; services
 1679 C and B have multiple top labels, but the respective services cannot ‘agree’ on
 1680 what the top label actually is. In Figure 4.3a, service C presents a reasonably high
 1681 confidence for the set of 7 top labels it returns, however there is too much diversity
 1682 ranging from a ‘hot chocolate’ to the hypernym ‘food’. Both are technically correct,
 1683 but it is up to the developer to decide the level of hypernymy to label the image as.
 1684 We also observe a similar effect in Figure 4.3b, where the image is labelled with
 1685 both the subject matter and the number of subjects per image.

1686 Thus, a taxonomy of ontologies is unknown; if a ‘border collie’ is detected in
 1687 an image, does this imply the hypernym ‘dog’ is detected, and then ‘mammal’, then
 1688 ‘animal’, then ‘object’? Only service B documents a taxonomy for capturing what
 1689 level of scope is desired, providing what it calls the ‘86-category’ concept as found
 1690 in its how-to guide:

1691 “*Identify and categorize an entire image, using a category taxonomy*
 1692 *with parent/child hereditary hierarchies. Categories can be used alone,*
 1693 *or with our new tagging models.”* [404]

1694 Thus, even if Rosa implemented conceptual similarity analysis for the image, the
 1695 top label set may not provide sufficient information to derive at a conclusive answer,
 1696 and if simply relying on only one label in this set, information such as the duplicity
 1697 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

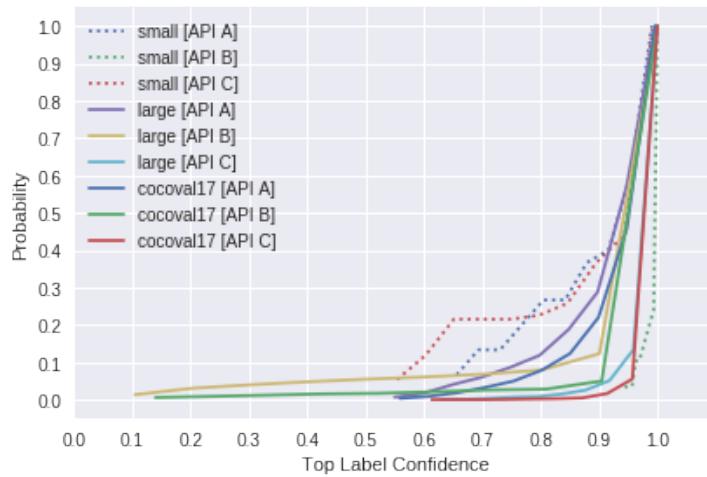


Figure 4.4: Cumulative distribution of the top labels' confidences. One in nine images return a top label(s) confident to $\gtrsim 97\%$, though there is a wider distribution for service A.

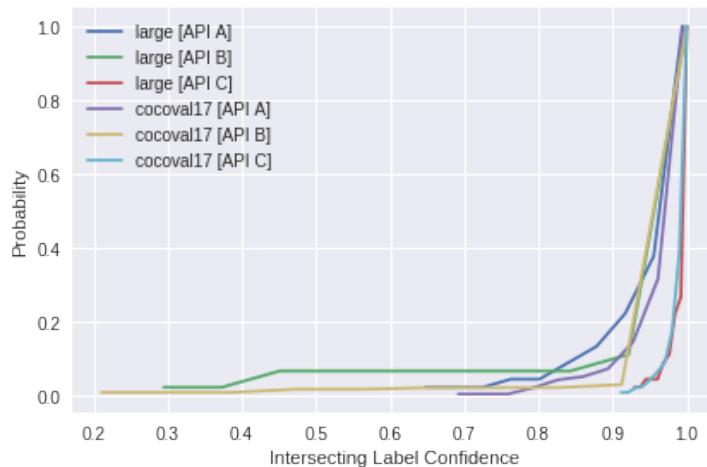


Figure 4.5: Cumulative distribution of intersecting top labels' confidences. The small data set is intentionally removed due to low intersections of labels (see Table 4.2).



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%.

¹⁷²⁸ deltas between snapshots (δ_c). However, this delta is so minuscule that we attribute
¹⁷²⁹ such changes to statistical noise.

¹⁷³⁰ 4.6 Recommendations

¹⁷³¹ 4.6.1 Recommendations for IWS users

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

¹⁷³³ Rosa should ensure that in her testing strategies for the app she develops, there is an
¹⁷³⁴ ontology focus for the types of vocabulary that are returned. Additionally, we noted
¹⁷³⁵ that there was a sudden change in case for services A and C; for all comparative
¹⁷³⁶ purposes of labels, each label should be lower-cased.

¹⁷³⁷ 4.6.1.2 *Incorporate a specialised IWS testing methodology into the development 1738 lifecycle*

¹⁷³⁹ Rosa can utilise the different aspects of consistency as outlined in this paper as
¹⁷⁴⁰ part of her quality strategy. To ensure results are correct over time, we recommend
¹⁷⁴¹ developers create a representative data set of the intended application’s data set
¹⁷⁴² and evaluate these changes against their chosen service frequently. This will help
¹⁷⁴³ identify when changes, if any, have occurred if vendors do not provide a line of
¹⁷⁴⁴ communication when this occurs.

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

¹⁷⁴⁶ Rosa will be locked into whichever vendor she chooses as there is inherent incon-
¹⁷⁴⁷ sistency between these services in both the vocabulary and ontologies that they use.
¹⁷⁴⁸ We have demonstrated that very few services overlap in their vocabularies, chiefly
¹⁷⁴⁹ because they are still in early development and there is yet to be an established,
¹⁷⁵⁰ standardised vocabulary that can be shared amongst the different vendors. Issues
¹⁷⁵¹ such as those shown in Section 4.5.1 can therefore be avoided.

1752 Throughout this work, we observed that the terminologies used by the vari-
1753 ous vendors are different. Documentation was studied, and we note that there is
1754 inconsistency between the ways techniques are described to users. We note the
1755 disparity between the terms ‘detection’, ‘recognition’, ‘localisation’ and ‘analysis’.
1756 This applies chiefly to object- and facial-related techniques. Detection applies to
1757 facial detection, which gives bounding box coordinates around all faces in an image.
1758 Similarly, localisation applies the same methodology to disparate objects in an image
1759 and labels them. In the context of facial ‘recognition’, this term implies that a face
1760 is *recognised* against a known set of faces. Lastly, ‘analysis’ applies in the context
1761 of facial analysis (gender, eye colour, expression etc.); there does not exist a similar
1762 analysis technique on objects.

1763 We notice similar patterns with object ‘tagging’, ‘detection’ and ‘labelling’.
1764 Service A uses ‘Entity Detection’ for object categorisation, service B uses ‘Image
1765 Tagging’, and service C uses the term ‘Detect Labels’ : conceptually, these provide
1766 the same functionality but the lack of consistency used between all three providers is
1767 concerning and leaves room for confusion with developers during any comparative
1768 analyses. Rosa may find that she wants to label her images into day/night scenes, but
1769 this in turn means the ‘labelling’ of varying objects. There is therefore no consistent
1770 standards to use the same terminology for the same concepts, as there are in other
1771 developer areas (such as Web Development).

1772 4.6.1.4 *Avoid use in safety-critical systems*

1773 We have demonstrated in this paper that both labels and confidences are stable but not
1774 constant; there is still an evolution risk posed to developers that may cause unknown
1775 consequences in applications dependent on these CVSs. Developers should avoid
1776 their use in safety critical systems due to the lack of visible changes.

1777 4.6.2 **Recommendations for IWS providers**

1778 4.6.2.1 *Improve the documentation*

1779 Rosa does not know that service A returns back ‘carrot’ for its top response, with
1780 service C returning ‘spoon’ (Figure 4.2b). She is unable to tell the service’s API
1781 where to focus on the image. Moreover, how can she toggle the level of specificity
1782 in her results? She is frustrated that service C can detect ‘chocolate’, ‘food’ and also
1783 ‘beverage’ all as the same top label in Figure 4.3a: what label is she to choose when
1784 the service is meant to do so for her, and how does she get around this? Thus, we
1785 recommend vendors to improve the documentation of services by making known
1786 the boundary set of the training data used for the algorithms. By making such
1787 information publicly available, developers would be able to review the service’s
1788 specificity for their intended use case (e.g., maybe Rosa is satisfied her app can
1789 catalogue ‘food’ together, and in fact does not want specific types of foods (‘hot
1790 chocolate’) catalogued). We also recommend that vendors publish usage guidelines
1791 should that include details of priors and how to evaluate the specific service results.

1792 Furthermore, we did not observe that the vendors documented how some images

1793 may respond with multiple labels of the exact same confidence value. It is not clear
1794 from the documentation that response objects can have duplicate top values, and
1795 tutorials and examples provided by the vendors do not consider this possibility. It
1796 is therefore left to the developer to decide which label from this top set of labels
1797 best suits for their particular use case; the documentation should describe that a rule
1798 engine may need to be added in the developer's application to verify responses. The
1799 implications this would have on maintenance would be significant.

1800 *4.6.2.2 Improve versioning*

1801 We recommend introducing a versioning system so that a model can be used from a
1802 specific date in production systems: when Rosa tests her app today, she would like
1803 the service to remain *static* the same for when her app is deployed in production
1804 tomorrow. Thus, in a request made to the vendor, Rosa could specify what date she
1805 ran her app's QA testing on so that she knows that henceforth these model changes
1806 will not affect her app.

1807 *4.6.2.3 Improve Metadata in Response*

1808 Much of the information in these services is reduced to a single confidence value
1809 within the response object, and the details about training data and the internal AI
1810 architecture remains unknown; little metadata is provided back to developers that
1811 encompass such detail. Early work into model cards and datasheets [124, 228]
1812 suggests more can be done to document attributes about ML systems, however at a
1813 minimum from our work, we recommend including a reference point via the form
1814 of an additional identifier. This identifier must also permit the developers to submit
1815 the identifier to another API endpoint should the developer wish to find further
1816 characteristics about the AI empowering the IWS, reinforcing the need for those
1817 presented in model cards and datasheets. For example, if Rosa sends this identifier
1818 she receives in the response object to the IWS descriptor API, she could find out
1819 additional information such as the version number or date when the model was
1820 trained, thereby resolving potential evolution risk, and/or the ontology of labels.

1821 *4.6.2.4 Apply constraints for predictions on all inputs*

1822 In this study, we used some images with intentionally disparate, and noisy objects. If
1823 services are not fully confident in the responses they give back, a form of customised
1824 error message should be returned. For example, if Rosa uploads an image of 10
1825 various objects on a table, rather than returning a list of top labels with varying
1826 confidences, it may be best to return a 'too many objects' exception. Similarly, if
1827 Rosa uploads a photo that the model has had no priors on, it might be useful to return
1828 an 'unknown object' exception than to return a label it has no confidence of. We do
1829 however acknowledge that current state of the art computer vision techniques may
1830 have limits in what they can and cannot detect, but this limitation can be exposed in
1831 the documentation to the developers.

1832 A further example is sending a one pixel image to the service, analogous to
1833 sending an empty file. When we uploaded a single pixel white image to service A,
1834 we received responses such as ‘microwave oven’, ‘text’, ‘sky’, ‘white’ and ‘black’
1835 with confidences ranging from 51–95%. Prior checks should be performed on all
1836 input data, returning an ‘insufficient information’ error where any input data is below
1837 the information of its training data.

1838 4.7 Threats to Validity

1839 4.7.1 Internal Validity

1840 Not all CVSSs were assessed. As suggested in Section 4.4, we note that there are
1841 other CVSSs such as IBM Watson. Many services from Asia were also not considered
1842 due to language barriers (of the authors) in assessing these services. We limited our
1843 study to the most popular three providers (outside of Asia) to maintain focus in this
1844 body of work.

1845 A custom confidence threshold was not set. All responses returned from each of
1846 the services were included for analysis; where confidences were low, they were still
1847 included for analysis. This is because we used the default thresholds of each API to
1848 hint at what real-world applications may be like when testing and evaluating these
1849 services.

1850 The label string returned from each service was only considered. It is common
1851 for some labels to respond back that are conceptually similar (e.g., ‘car’ vs. ‘automobile’)
1852 or grammatically different (e.g., ‘clothes’ vs. ‘clothing’). While we could have
1853 employed more conceptual comparison or grammatical fixes in this study, we chose
1854 only to compare lowercased labels and as returned. We leave semantic comparison
1855 open to future work.

1856 Only introductory analysis has been applied in assessing the documentation of
1857 these services. Further detailed analysis of documentation quality against a rigorous
1858 documentation quality framework would be needed to fortify our analysis of the
1859 evolution of these services’ documentation.

1860 4.7.2 External Validity

1861 The documentation and services do change over time and evolve, with many allowing
1862 for contributions from the developer community via GitHub. We note that our
1863 evaluation of the documentation was conducted on a single date (see Section 4.4)
1864 and acknowledge that the documentation may have changed from the evaluation date
1865 to the time of this publication. We also acknowledge that the responses and labelling
1866 may have evolved too since the evaluation period described and the date of this
1867 publication. Thus, this may have an impact on the results we have produced in this
1868 paper compared to current, real-world results. To mitigate this, we have supplied the
1869 raw responses available online [381].

1870 Moreover, in this paper we have investigated *computer vision* services. Thus,
1871 the significance of our results to other domains such as natural language processing

1872 or audio transcription is, therefore, unknown. Future studies may wish to repeat our
1873 methodology on other domains to validate if similar patterns occur; we remain this
1874 open for future work.

1875 4.7.3 Construct Validity

1876 It is not clear if all the recommendations proposed in Section 4.6 are feasible
1877 or implementable in practice. Construct validity defines how well an experiment
1878 measures up to its claims; the experiments proposed in this paper support our three
1879 hypotheses but these have been conducted in a clinical condition. Real-world case
1880 studies and feedback from developers and providers in industry would remove the
1881 controlled nature of our work.

1882 4.8 Conclusions & Future Work

1883 This study explored three popular CVSs over an 11 month longitudinal experiment
1884 to determine if these services pose any evolution risk or inconsistency. We find that
1885 these services are generally stable but behave inconsistently; responses from these
1886 services do change with time and this is not visible to the developers who use them.
1887 Furthermore, the limitations of these systems are not properly conveyed by vendors.
1888 From our analysis, we present a set of recommendations for both IWS vendors and
1889 developers.

1890 Standardised software quality models (e.g., [159]) target maintainability and
1891 reliability as primary characteristics. Quality software is stable, testable, fault
1892 tolerant, easy to change and mature. These CVSs are, however, in a nascent stage,
1893 difficult to evaluate, and currently are not easily interchangeable. Effectively, the
1894 IWS response objects are shifting in material ways to developers, albeit slowly, and
1895 vendors do not communicate this evolution or modify API endpoints; the endpoint
1896 remains static but the content returned does not despite the same input.

1897 There are many potential directions stemming from this work. To start, we plan
1898 to focus on preparing a more comprehensive datasheet specifically targeted at what
1899 should be documented to application developers, and not data scientists. Reapplying
1900 this work in real-world contexts, that is, to get real developer opinions and study
1901 production grade systems, would also be beneficial to understand these phenomena
1902 in-context. This will help us clarify if such changes are a real concern for developers
1903 (i.e., if they really need to change between services, or the service evolution has real
1904 impact on their applications). We also wish to refine and systematise the method
1905 used in this study and develop change detectors that can be used to identify evolution
1906 in these services that can be applied to specific ML domains (i.e., not just computer
1907 vision), data sets, and API endpoints, thereby assisting application developers in their
1908 testing strategies. Moreover, future studies may wish to expand the methodology
1909 applied by refining how the responses are compared. As there does not yet exist a
1910 standardised list of terms available between services, labels could be *semantically*
1911 compared instead of using exact matches (e.g., by using stem words and synonyms
1912 to compare similar meanings of these labels), similar to previous studies [245].

1913 This paper has highlighted only some high-level issues that may be involved
1914 in using these evolving services. The laws of software evolution suggest that for
1915 software to be useful, it must evolve [224, 325]. There is, therefore, a trade-off, as
1916 we have shown, between consistency and evolution in this space. For a component
1917 to be stable, any changes to dependencies it relies on must be communicated. We
1918 are yet to see this maturity of communication from IWS providers. Thus, developers
1919 must be cautious between integrating intelligent components into their applications
1920 at the expense of stability; as the field of AI is moving quickly, we are more likely to
1921 see further instability and evolution in IWSs as a consequence.

CHAPTER 5

1922

1923

1924

Interpreting Pain-Points in Computer Vision Services[†]

1925

1926 **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

1946 The availability of recent advances in artificial intelligence (AI) over simple RESTful
1947 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: IEEE, October 2020, In Press. Terminology has been updated to fit this thesis.

1948 web services (IWSs) are AI components that abstract complex machine learning
1949 (ML) and AI techniques behind simpler API calls. In particular, they hide (either
1950 explicitly or implicitly) any data-driven and non-deterministic properties inherent
1951 to the process of their construction. The promise is that software engineers can
1952 incorporate complex machine learnt capabilities, such as computer vision, by simply
1953 calling an API end-point.

1954 The expectation is that application developers can use these AI-powered services
1955 like they use other conventional software components and cloud services (e.g., object
1956 storage like AWS S3). Furthermore, the documentation of these AI components is
1957 still anchored to the traditional approach of briefly explaining the end-points with
1958 some information about the expected inputs and responses. The presupposition
1959 is that developers can reason and work with this high level information. These
1960 services are also marketed to suggest that application developers do not need to fully
1961 understand how these components were created (i.e., assumptions in training data
1962 and training algorithms), the ways in which the components can fail, and when such
1963 components should and should not be used.

1964 The nuances of ML and AI powering IWSs have to be appreciated, as there are
1965 real-world consequences to software quality for applications that depend on them if
1966 they are ignored [81]. This is especially true when ML and AI are abstracted and
1967 masked behind a conventional-looking API call, yet the mechanisms behind the API
1968 are data-dependent, probabilistic and potentially non-deterministic [245]. We are
1969 yet to discover what long-term impacts exist during development and production due
1970 to poor documentation that do not capture these traits, nor do we know the depth of
1971 understanding application developers have for these components. Given the way AI-
1972 powered services are currently presented, developers are also likely to reason about
1973 these new services much like a string library or a cloud data storage service. That
1974 is, they may not fully consider the implications of the underlying statistical nature
1975 of these new abstractions or the consequent impacts on productivity and quality.

1976 Typically, when developers are unable to correctly align to the mindset of the
1977 API designer, they attempt to resolve issues by (re-)reading the API documentation.
1978 If they are still unable to resolve these issues on their own after some internet
1979 searching, they consider online discussion platforms (e.g., Stack Overflow, GitHub
1980 Issues, Mailing Lists) where they seek technological advice from their peers [3].
1981 Capturing what developers discuss on these platforms offers an insight into the
1982 frustrations developers face when using different software components as shown
1983 by recent works [33, 175, 283, 311, 339]. However, to our knowledge, no studies
1984 have yet analysed what developers struggle with when using the new generation of
1985 *intelligent* services. Given the re-emergent interest in AI and the anticipated value
1986 from this technology [205], a better understanding of issues faced by developers
1987 will help us improve the quality of services. Our hypothesis is that application
1988 developers do not fully appreciate the probabilistic nature of these services, nor do
1989 they have sufficient appreciation of necessary background knowledge—however, we
1990 do not know the specific areas of concern. The motivation for our study is to inform
1991 API designers on which aspects to focus in their documentation, education, and
1992 potentially refine the design of the end-points.

1993 This study involves an investigation of 1,825 Stack Overflow (SO) posts regarding
1994 one of the most mature types of IWSs—computer vision services (CVSs)—dating
1995 from November 2012 to June 2019. We adapt existing methodologies of prior SO
1996 analyses [33, 320] to extract posts related to CVSs. We then apply two existing SO
1997 question classification schemes presented at ICPC and ICSE in 2018 and 2019 [3, 34].
1998 These previous studies focused on mobile apps and web applications. Although not
1999 a direct motivation, our work also serves as a validation of the applicability of these
2000 two issue classification taxonomies [3, 34] in the context of IWSs (hence potential
2001 for generalisation). Additionally our work is the first—to our knowledge—to *test*
2002 the applicability of these taxonomies in a new study.

2003 The taxonomies in previous works focus on the specific aspects from the domain
2004 (e.g. API usage, specificity within the documentation etc.) and as such do not
2005 deeply consider the learning gap of an application developer. To explore the API
2006 learning implications raised by our SO analysis, we applied an additional lens of
2007 two taxonomies from the field of pedagogy. This was motivated by the need to offer
2008 an insight into the work needed to help developers learn how to use these relatively
2009 new services.

2010 The key findings of our study are:

- 2011 • The primary areas that developers raise as issues reflect a relatively primitive
2012 understanding of the underlying concepts of data-driven ML approaches used.
2013 We note this via the issues raised due to conceptual misunderstanding and
2014 confusion in interpreting errors,
- 2015 • Developers predominantly encounter a different distribution of issue types than
2016 were reported in previous studies, indicating the complexity of the technical
2017 domain has a non-trivial influence on intelligent API usage; and
- 2018 • Most of these issues can be resolved with better documentation, based on our
2019 analysis.

2020 The paper also offers a data-set as an additional contribution to the research
2021 community and to permit replication [382]. The paper structure is as follows:
2022 Section 5.2 provides motivational examples to highlight the core focus of our study;
2023 Section 5.3 provides a background on prior studies that have mined SO to gather
2024 insight into the software engineering community; Section 5.4 describes our study
2025 design in detail; Section 5.5 presents the findings from the SO extraction; Section 5.6
2026 offers an interpretation of the results in addition to potential implications that arise
2027 from our work; Section 5.7 outlines the limitations of our study; concluding remarks
2028 are given in Section 5.8.

2029 **5.2 Motivation**

2030 “Intelligent” services are often available as a cloud end-point and provide devel-
2031 opers a friendly approach to access recent AI/ML advances without being experts
2032 in the underlying processes. Figure 5.1 highlights how these services abstract
2033 away much of the technical know-how needed to create and operationalise these
2034 IWSs [248]. In particular, they hide information about the training algorithm and

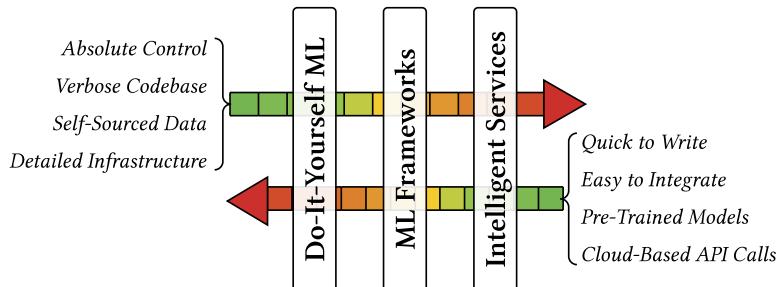


Figure 5.1: Some traits of Intelligent Services vs. ‘Do-It-Yourself’ ML. Green-to-red arrows indicate the presence of these traits. *Adapted from Ortiz [248].*

2035 data-sets used in training, the evaluation procedures, the optimisations undertaken,
2036 and—surprisingly—they often do not offer a properly versioned end-point [81, 245].
2037 That is, the cloud vendors may change the behaviour of the services without sufficient
2038 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.” [426]; “I’m trying out the [IWS], and there’s a score field that returns that I’m not sure how to interpret [it].” [440]
 - **frustrated about non-deterministic results:** “Often the API has troubles in recognizing single digits... At other times Vision confuses digits with letters.” [439]; “Is there a way to help the program recognize numbers better, for example limit the results to a specific format, or to numbers only?” [436]
 - **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.” [420]
 - **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.” [423]

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 [3, 34, 320]. 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:

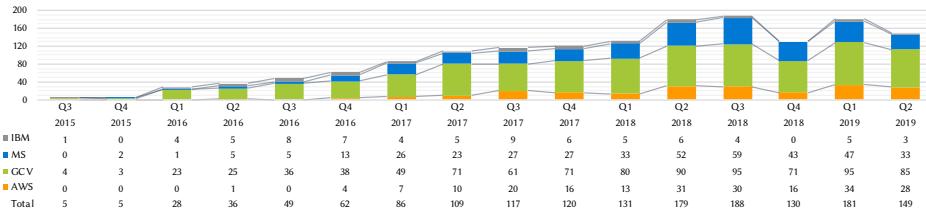


Figure 5.2: Trend of posts, where IBM = IBM Watson Visual Recognition, MS = Azure Computer Vision, AWS = AWS Rekognition and GCV = Google Cloud Vision. Three MS posts from Q4 2012, Q3 2013 and Q4 2013 have been removed for graph clarity.

- 2067 **RQ1. How do developers mis-comprehend IWSs as presented within SO**
 2068 **pain-points?** While the AI community is well aware in the the nuances that
 2069 empower IWSs, such services are being released for application developers
 2070 who may not be aware of their limitations or how they work. This is
 2071 especially the case when machine intelligence is accessed via web-based
 2072 APIs where such details are not fully exposed.
- 2073 **RQ2. Are the distribution of issues similar to prior studies?** We compare
 2074 how the distributions of previous studies' of posts about conventional,
 2075 deterministic API services differ from those of IWSs. By assessing the
 2076 distribution of IWSs' issues against similar studies that focus on mobile
 2077 and web development, we identify whether a new taxonomy is needed
 2078 specific to AI-based services, and if gaps specific to AI knowledge exist
 2079 that need to be captured in these taxonomies.

2080 5.3 Background

- 2081 The primary goal of analysing issues is to better understand the root causes. Hence,
 2082 a good issue classification taxonomy should ideally capture the underlying causal
 2083 aspects (instead of pure functional groupings) [70]. Although this idea (of cause
 2084 related classification) is not new (Chillarege advocated for it in this TSE paper in
 2085 1992), this is not a universally followed approach when studying online discussions
 2086 and some recent works have largely classified issues into the “*what is*” and not
 2087 “*how to fix it*” [23, 33, 328]. They typically (manually) classify discussion into
 2088 either *functional areas* (e.g., Website Design/CSS, Mobile App Development, .NET
 2089 Framework, Java [23]) or *descriptive areas* (e.g., Coding Style/Practice, Problem/-
 2090 Solution, Design, QA [23, 328]). As a result, many of these studies do not give
 2091 us a prioritised means of targeted attack on how to *resolve* these issues with, for
 2092 example, improved documentation. Interestingly, recent taxonomies that studied SO
 2093 data (Aghajani et al. [3] and Beyer et al. [34]) were causal in nature and developed to
 2094 understand discussions related to mobile and web applications. However, issues that
 2095 arise when developers use IWSs have not been studied, nor do we know if existing
 2096 issue classification taxonomies are sufficient in this domain.
- 2097 Researchers studying APIs have also attempted to understand developer's opin-
 2098 ions towards APIs [328], categorise the questions they ask about these APIs [23,

2099 25, 34, 283], and understand API related documentation and usage issues [3, 4, 7,
2100 23, 150, 320]. These studies often employ automation to assist in the data analysis
2101 stages of their research. Latent Dirichlet Allocation [7, 23, 283, 328] is applied for
2102 topic modelling and other ML techniques such as Random Forests [34], Conditional
2103 Random Fields [4] or Support Vector Machines [34, 150] are also used.

2104 However, automatic techniques are tuned to classify into *descriptive* categories,
2105 that is, they help paint a landscape of *what is*, but generally do not address the
2106 causal factors to address the issues in great detail. For example, functional areas
2107 such as ‘Website Design’ [23], ‘User Interface’ [33] or ‘Design’ [329] result from
2108 such analyses. These automatic approaches are generally non-causal, making it hard
2109 to address reasons for *why* developers are asking such questions. However, not all
2110 studies in the space use automatic techniques; other studies employ manual thematic
2111 analysis [3, 25, 320] (e.g., card sorting) or a combination of both [33, 34, 283, 327].
2112 Our work uses a manual approach for classification, and we use taxonomies that
2113 are more causally aligned allowing our findings to be directly useful in terms of
2114 addressing the issues.

2115 Evidence-based software engineering [180] has helped shape the last 15 years
2116 worth of research, but the reliability of such evidence has been questioned [169, 171,
2117 302]. Replication studies, especially in empirical works, can give us the confidence
2118 that existing results are adaptable to new domains; in this context, we extend (to
2119 IWSs) and work with study methods developed in previous works.

2120 5.4 Method

2121 5.4.1 Data Extraction

2122 This study initially attempted to capture SO posts on a broad range of many IWSs by
2123 identifying issues related to four popular IWS cloud providers: Google Cloud [388],
2124 AWS [363], Azure [402] and IBM Cloud [398]. We based our selection criteria on
2125 the prominence of the providers in industry (Google, Amazon, Microsoft, IBM) and
2126 their ubiquity in cloud platform services. Additionally, in 2018, these services were
2127 considered the most adopted cloud vendors for enterprise applications [277].

2128 However, during the filtering stage (see Section 5.4.2), we decided to focus on
2129 a subset of these services, computer vision, as these are one of the more mature
2130 and stable ML/AI-based services with widespread and increasing adoption in the
2131 developer community (see Figure 5.2). We acknowledge other services beyond the
2132 four analysed provide similar capabilities [376, 377, 384, 397, 449, 450] and only
2133 English-speaking services have been selected, excluding popular services from Asia
2134 (e.g., [374, 375, 396, 415, 416])—see Section 5.7. For comprehensiveness, we
2135 explain below our initial attempts to extract *all* IWSs.

2136 5.4.1.1 Defining a list of IWSs

2137 As there exists no global ‘list’ of IWSs to search on, we needed to derive a *corpus*
2138 of *initial terms* to allow us to know *what* to search for on the Stack Exchange Data

2139 Explorer¹ (SEDE). We began by looking at different brand names of cloud services
2140 and their permutations (e.g., Google Cloud Services and GCS) as well as various
2141 ML-related products (e.g., Google Cloud ML). To do this, we performed extensive
2142 Google searches² in addition to manually reviewing six ‘overview’ pages of the
2143 relevant cloud platforms. We identified 91 initial IWSs to incorporate into our
2144 search terms³.

2145 5.4.1.2 *Manual search for relevant, related terms*

2146 We then ran a manual search² on each term to determine if these terms were relevant.
2147 We did this by querying each term within SO’s search feature, reviewing the titles
2148 and body post previews of the first three pages of results (we did not review the
2149 answers, only the questions). We also noted down the user-defined *Tags* of each post
2150 (up to five per question); by clicking into each tag, we could review similar tags (e.g.,
2151 ‘project-oxford’ for ‘azure-cognitive-services’) and check if the tag had synonyms
2152 (e.g., ‘aws-lex’ and ‘amazon-lex’). We then compiled a *corpus of tags* consisting of
2153 31 terms.

2154 5.4.1.3 *Developing a search query*

2155 We recognise that searching SEDE via *Tags* exclusively can be ineffective (see [23,
2156 320]). To mitigate this, we produced a *corpus of title and body terms*. Such terms
2157 are those that exist within the title and body of the posts to reflect the ways in
2158 which individual developers commonly use to refer to different IWSs. To derive
2159 at such a list, we performed a search^{2,3} of the 31 tags above in SEDE, filtering
2160 out posts that were not answers (i.e., questions only) as we wanted to see how
2161 developers *phrase* their questions. For each search, we extracted a random sample
2162 of 100 questions (400 total for each service) and reviewed each question. We noted
2163 many patterns in the permutations of how developers refer to these services, such
2164 as: common misspellings ('bind' vs. 'bing'); brand misunderstanding ('Microsoft
2165 computer vision' vs. 'Azure computer vision'); hyphenation ('Auto-ML' vs. 'Auto
2166 ML'); UK and US English ('Watson Analyser' vs. 'Watson Analyzer'); and, the
2167 use of apostrophes, plurals, and abbreviations ('Microsoft's Computer Vision API',
2168 'Microsoft Computer Vision Services', 'GCV' vs. 'Google Cloud Vision'). We
2169 arrived at a final list of 229 terms compromising all of the IWSs provided by
2170 Google, Amazon, Microsoft and IBM as of January 2019³.

2171 5.4.1.4 *Executing our search query*

2172 Our next step was to perform a case-insensitive search of all 229 terms within the
2173 body or title of posts. We used Google BigQuery’s public data-set of SO posts⁴ to
2174 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>

²¹⁷⁵ search was conducted on 10 May 2019, where we extracted 21,226 results. We then
²¹⁷⁶ performed several filtering steps to cleanse our extracted data, as explained below.

²¹⁷⁷ 5.4.2 Data Filtering

²¹⁷⁸ 5.4.2.1 Refining our inclusion/exclusion criteria

²¹⁷⁹ We performed an initial manual filtering of the 50 most recent posts (sorted by
²¹⁸⁰ descending *CreationDate* values) of the 21,226 posts above, assessing the suitability
²¹⁸¹ of the results and to help further refine our inclusion and exclusion criteria. We
²¹⁸² did note that some abbreviations used in the search terms (e.g., ‘GCV’, ‘WCS’⁵),
²¹⁸³ resulting in irrelevant questions in our result set. We therefore removed abbreviations
²¹⁸⁴ from our search query and consolidated all overlapping terms (e.g., ‘Google Vision
²¹⁸⁵ API’ was collapsed into ‘Google Vision’).

²¹⁸⁶ We also recognised that 21,226 results would be non-trivial to analyse without
²¹⁸⁷ automated techniques. As we wanted to do manual qualitative analysis, we reduced
²¹⁸⁸ our search space to 27 search terms of just the CVSs within the original corpus of
²¹⁸⁹ 229 terms. These were Google Cloud Vision [388], AWS Rekognition [363], Azure
²¹⁹⁰ Computer Vision [402], and IBM Watson Visual Recognition [398]. This resulted
²¹⁹¹ in 1,425 results that were extracted on 21 June 2019. The query used and raw results
²¹⁹² are available online in our supplementary materials [382].

²¹⁹³ 5.4.2.2 Duplicates

²¹⁹⁴ Within 1,425 results, no duplicate questions were noted, as determined by unique
²¹⁹⁵ post ID, title or timestamp.

²¹⁹⁶ 5.4.2.3 Automated and manual filtering

²¹⁹⁷ To assess the suitability and nature of the 1,425 questions extracted, the first author
²¹⁹⁸ began with a manual check on a randomised sample of 50 questions. As the questions
²¹⁹⁹ were exported in a raw CSV format (with HTML tags included in the post’s body), we
²²⁰⁰ parsed the questions through an ERB templating engine script⁶ in which the ID, title,
²²⁰¹ body, tags, created date, and view, answer and comment counts were rendered for
²²⁰² each post in an easily-readable format. Additionally, SQL matches in the extraction
²²⁰³ process were also highlighted in yellow (i.e., in the body of the post) and listed at
²²⁰⁴ the top of each post. These visual cues helped to identify 3 false positive matches
²²⁰⁵ where library imports or stack traces included terms within our corpus of 26 CVS
²²⁰⁶ terms. For example, `aws-java-sdk-rekognition:jar` is falsely matched as a
²²⁰⁷ dependency within an unrelated question. As such exact matches would be hard to
²²⁰⁸ remove without the use of regular expressions, and due to the low likelihood (6%)
²²⁰⁹ 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>

2210 **5.4.2.4 Classification**

2211 Our 1,425 posts were then split into 4 additional random samples (in addition to the
2212 random sample of 50 above). 475 posts were classified by the first author and three
2213 other research assistants, software engineers with at least 2 years industry experience,
2214 assisted to classify the remaining 900. This left a total of 1,375 classifications
2215 made by four people plus an additional 450 classifications made from reliability
2216 analysis, in which the remaining 50 posts were classified nine times (as detailed in
2217 Section 5.4.3.1). Thus, a total of 1,825 classifications were made from the original
2218 1,425 posts extracted.

2219 Whilst we could have chosen to employ topic modelling, these are too descriptive
2220 in nature (as discussed in Section 5.3). Moreover, we wanted to see if prior
2221 taxonomies can be applied to IWSs (as opposed to creating a new one) and compare
2222 if their distributions are similar. Therefore, we applied the two existing taxonomies
2223 described in Section 5.3 to each post; (i) a documentation-specific taxonomy that ad-
2224 dresses issues directly resulting from documentation, and (ii) a generalised taxonomy
2225 that covers a broad range of SO issues in a well-defined software engineering area
2226 (specifically mobile app development). Aghajani et al.'s documentation-specific
2227 taxonomy (Taxonomy A) is multi-layered consisting of four dimensions and 16
2228 sub-categories [3]. Similarly, Beyer's SO generalised post classification taxonomy
2229 (Taxonomy B) consists of seven dimensions [34]. We code each dimension with
2230 a number, X , and each sub-category with a letter y : (Xy). We describe both tax-
2231 onomies in detail within Table 5.1. Where a post was included in our results but
2232 not applicable to IWSs (see Section 5.4.2.3) or not applicable to a taxonomy dimen-
2233 sion/category, then the post was flagged for removal in further analysis. Table 5.1
2234 presents *our understanding* of the respective taxonomies; our intent is not to method-
2235 logically replicate Aghajani et al. or Beyer et al.'s studies in the IWS domain, rather
2236 to acknowledge related work in the area of SO classification and reduce the need to
2237 synthesise a new taxonomy. We baseline all coding against *our interpretation only*.
2238 Our classifications are therefore independent of the previous authors' findings.

2239 **5.4.3 Data Analysis**

2240 **5.4.3.1 Reliability of Classification**

2241 To measure consistency of the categories assigned by each rater to each post, we
2242 utilised both intra- and inter-rater reliability [218]. As verbatim descriptions from
2243 dimensions and sub-categories were considered quite lengthy from their original
2244 sources, all raters met to agree on a shared interpretation of the descriptions, which
2245 were then paraphrased as discussed in the previous subsection and tabulated in
2246 Table 5.1. To perform statistical calculations of reliability, each category was as-
2247 signed a nominal value and a random sample of 50 posts were extracted. Two-phase
2248 reliability analysis followed.

2249 Firstly, intra-rater agreement by the first author was conducted twice on 28 June
2250 2019 and 9 August 2019. Secondly, inter-rater agreement was conducted with the
2251 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. [3])	
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. [34])	
B-1	■ API usage
B-2	■ Discrepancy
B-3	■ Errors
B-4	■ Review
B-5	■ Conceptual
B-6	■ API change
B-7	■ Learning

2252 group in mid-August 2019. Thus, the 50 posts were classified an additional nine
2253 times, resulting in 450 classifications for reliability analysis. We include these
2254 classifications in our overall analysis.

2255 At first, we followed methods of reliability analysis similar to previous SO
2256 studies (e.g., [320]) using the percentage agreement metric that divides the number
2257 of agreed categories assigned per post by the total number of raters [218]. However,
2258 percentage agreement is generally rejected as an inadequate measure of reliability
2259 analysis [75, 137, 186] in statistical communities. As we used more than 2 coders
2260 and our reliability analysis was conducted under the same random sample of 50
2261 posts, we applied *Light's Kappa* [197] to our ratings, which indicates an overall
2262 index of agreement. This was done using the `irr` computational R package [119]
2263 as suggested in [137].

2264 **5.4.3.2 Distribution Analysis**

2265 In order to compare the distribution of categories from our study with previous studies
2266 we carried out a χ^2 test. We selected a χ^2 test as the following assumptions [303]
2267 are satisfied: (i) the data is categorical, (ii) all counts are greater than 5, and (iii)
2268 we can assume simple random sampling. The null hypothesis describes the case
2269 where each population has the same proportion of observations and the alternative
2270 hypothesis is where at least one of the null hypothesis statements is false. We chose
2271 a significance value, α , of 0.05 following a standard rule of thumb. As to the best
2272 of our knowledge this is the first statistical comparison using Taxonomy A and B on
2273 SO posts. To report the effect size we selected Cramer's Phi, ϕ_c which is well suited
2274 for use on nominal data [303].

2275 **5.5 Findings**

2276 We present our findings from classifying a total of 1,825 SO posts aimed at answering
2277 RQs 1 and 2. 450 posts were classified using Taxonomies A and B for reliability
2278 analysis as described in Section 5.4.3.1 and the remaining 1,375 posts were classified
2279 as per Section 5.4.2.4. A summary of our classification using Taxonomies A and B
2280 is shown in Figure 5.3.

2281 **5.5.1 Post classification and reliability analysis**

2282 When undertaking the classification, we found that 238 issues (13.04%) did not
2283 relate to IWSs directly. For example, library dependencies were still included in
2284 a number of results (see Section 5.4.2.3), and we found there to be many posts
2285 discussing Android's Mobile Vision API as Google (Cloud) Vision. These issues
2286 were flagged and ignored for further analysis (see Section 5.4.2.4).

2287 For our reliability analysis, we classified a total of 450 posts of which 70 posts
2288 were flagged as irrelevant. Landis and Koch [192] provide guidelines to interpret
2289 kappa reliability statistics, where $0.00 \leq \kappa \leq 0.20$ indicates *slight* agreement and
2290 $0.21 \leq \kappa \leq 0.40$ indicates *fair* agreement. Despite all raters meeting to agree
2291 on a shared interpretation of the taxonomies (see Section 5.4.3.1) our inter-rater

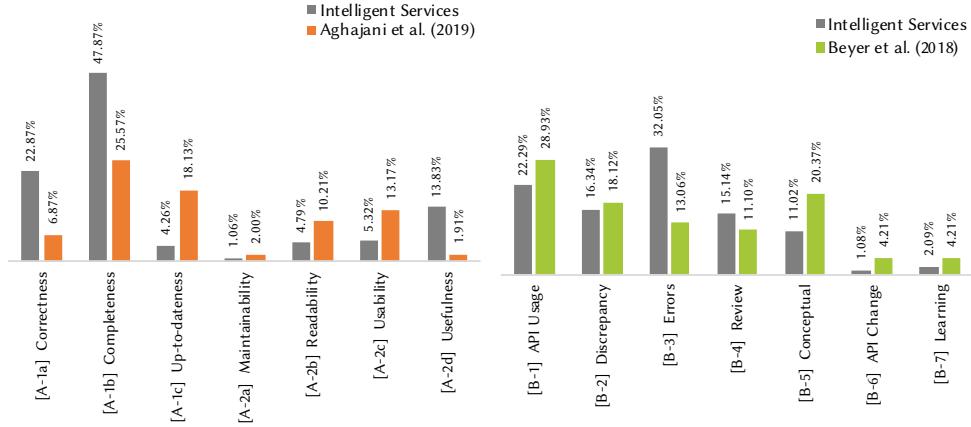


Figure 5.3: *Left:* Documentation-specific classification taxonomy results highlights a mostly similar distribution to that of Aghajani et al.’s findings [3]. *Right:* Generalised classification taxonomy results highlight differences from more mature fields (i.e., Android APIs in Beyer et al. [34]) to 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?*” [423]; “*There seems to be very little to no documentation for AWS iOS text recognition inside an image*” [421].

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

2316 mentation, especially in relation to incorrect quotas and limitations: “*Are the Cloud*
2317 *Vision API limits in documentation correct?*” [434], “*According to the Google Vision*
2318 *documentation, the maximum number of image files per request is 16. Elsewhere,*
2319 *however, I’m finding that the maximum number of requests per minute is as high as*
2320 *1800.*” [419].

2321 There are also many references (23.94%) addressing the confusing nature of
2322 some documentation, indicating that the **readability, usability and usefulness of**
2323 **the documentation (A-2b, A-2c and A-2d)** could be improved. For example, “*Am*
2324 *I encoding it correctly? The docs are quite vague.*” [417], “*The aws docs for this*
2325 *are really confusing.*” [446].

2326 5.5.2.2 *Results from Beyer et al.’s taxonomy*

2327 We found that a majority (32.05%) of posts are primarily **error-related questions**
2328 **(B-3)**, including a dump of the stack trace or exception message from the service’s
2329 programming-language SDK (usually Java, Python or C#) that relates to a specific
2330 error. For example: “*I can’t fix an error that’s causing us to fall behind.*” [443]; “*I’m*
2331 *using the Java Google Vision API to run through a batch of images... I’m now getting*
2332 *a channel closed and ClosedChannelException error on the request.*” [437].

2333 **API usage questions (B-1)** were the second highest category at 22.29% of
2334 posts. Reading the questions revealed that many developers present an insufficient
2335 understanding of the behaviour, functional capability and limitation of these services
2336 and the need for further data processing. For example, while Azure provides an
2337 image captioning service, this is not universal to all CVSSs: “*In Amazon Rekognition*
2338 *for image processing how do I get the caption for an image?*” [428]. Similarly,
2339 OCR-related and label-related questions often indicate interest in cross-language
2340 translation, where a separate translation service would be required: “*Can Google*
2341 *Cloud Vision generate labels in Spanish via its API?*” [442]; “[*How can I] specify*
2342 *language for response in Google Cloud Vision API*” [429]; “*When I request a text*
2343 *detection of an image, it gives only English Alphabet characters (characters without*
2344 *accents) which is not enough for me. How can I get the UTF-32 characters?*” [424].

2345 It was commonplace to see questions that demonstrate a lack of depth in under-
2346 standing and appreciating how these services work, instead posting simple debugging
2347 questions. For instance, in the 11.02% of **conceptual-related questions (B-5)** that
2348 we categorised, we noticed causal links to a misunderstanding (or lack of aware-
2349 ness) of the vocabulary used within computer vision. For example: “*The problem*
2350 *is that I need to know not only what is on the image but also the position of that*
2351 *object. Some of those APIs have such feature but only for face detection.*” [435];
2352 “*I want to know if the new image has a face similar to the original image.... [the*
2353 *service] can identify faces, but can I use it to get similar faces to the identified face*
2354 *in other images?*” [427]. It is evident that some application developers are not aware
2355 of conceptual differences in computer vision such as object/face *detection* versus
2356 *localisation* versus *recognition*.

2357 In the 16.34% of **discrepancy-related questions (B-2)**, we see further unaware-
2358 ness from developers in how the underlying systems work. In OCR-related questions,
2359 developers do not understand the pre-processing steps required before an OCR is

2360 performed. In instances where text is separated into multiple columns, for example,
 2361 text is read top-down rather than left-to-right and segmentation would be required
 2362 to achieve the expected results. For example, “*it appears that the API is using some*
 2363 *kind of logic that makes it scan top to bottom on the left side and moving to right*
 2364 *side and doing a top to bottom scan.*” [441]; “*this method returns scanned text in*
 2365 *wrong sequence... please tell me how to get text in proper sequence.*” [447].

2366 A number of **review-related questions (B-4)** (15.14%) seem to provide some
 2367 further depth in understanding the context to which these systems work, where train-
 2368 ing data (or training stages) are needed to understand how inferences are made: “*How*
 2369 *can we find an exhaustive list (or graph) of all logos which are effectively recognized*
 2370 *using Google Vision logo detection feature?*” [445]; “*when object banana is detected*
 2371 *with accuracy greater than certain value, then next action will be dispatched... how*
 2372 *can I confidently define and validate the threshold value for each item?*” [431].

2373 **API change (B-6)** was shown in 1.08% of posts, with evolution of the services
 2374 occurring (e.g., due to new training data) but not necessarily documented “*Recently*
 2375 *something about the Google Vision API changed... Suddenly, the API started to*
 2376 *respond differently to my requests. I sent the same picture to the API today, and I*
 2377 *got a different response (from the past).*” [444].

2378 5.5.3 Statistical Distribution Analysis

2379 We obtained the following results $\chi^2 = 131.86$, $\alpha = 0.05$, $p \text{ value} = 2.2 \times 10^{-16}$ and
 2380 $\phi_c = 0.362$ from our distribution analysis with Taxonomy A to compare our study
 2381 with that of Aghajani et al. [3]. Comparing our study to Beyer et al. [34] produced the
 2382 following results $\chi^2 = 145.58$, $\alpha = 0.05$, $p \text{ value} = 2.2 \times 10^{-16}$ and $\phi_c = 0.252$.
 2383 These results show that we are able to reject the null hypothesis that the distribution
 2384 of posts using each taxonomy was the same as the comparison study. While there are
 2385 limited guidelines for interpreting ϕ_c when there is no prior information for effect
 2386 size [315], Sun et al. suggests the following: $0.07 \leq \phi_c \leq 0.20$ indicates a *small*
 2387 effect, $0.21 \leq \phi_c \leq 0.35$ indicates a *medium* effect, and $0.35 > \phi_c$ indicates a *large*
 2388 effect. Based on this criteria we obtained a *large* effect size for the documentation-
 2389 specific classification (Taxonomy A) and a *medium* effect size for the generalised
 2390 classification (Taxonomy B).

2391 5.6 Discussion

2392 5.6.1 Answers to Research Questions

2393 5.6.1.1 *How do developers mis-comprehend IWSs as presented within SO pain-*
 2394 *points? (RQ1)*

2395 Upon meeting to discuss the discrepancies between our categorisation of IWS usage
 2396 SO posts, we found that our interpretations of the *posts themselves* were largely sub-
 2397 jective. For example, many posts presented multi-faceted dimensions for Taxonomy
 2398 B; Beyer et al. [34] argue that a post can have more than one question category and

2399 therefore multi-label classification is appropriate at times. We highlight this further
2400 in the threats to validity (Section 5.7).

2401 We have to define the context of IWSs to address RQ1. We use the concept
2402 of a “technical domain” [20] to define this context. A technical domain captures
2403 the domain-specific concerns that influence the non-functional requirements of a
2404 system [20]. In the context of IWSs, the technical domain includes exploration, data
2405 engineering, distributed infrastructure, training data, and model characteristics as
2406 first class citizens [20]. We would then expect to see posts on SO related to these
2407 core concerns.

2408 In Figure 5.3, for the documentation-specific classification, the majority of posts
2409 were classified as **Completeness (A1-b)** related (47.87%). An interpretation for this
2410 is that the documentation does not adequately cover the technical domain concerns.
2411 Comments by developers such as “*I'm searching for a list of all the possible image*
2412 *labels that the Google Cloud Vision API can return?*” [422] indicates the documen-
2413 *tation does not adequately describe the training data for the API—developers do*
2414 *not know the required usage assumptions. Another quote from a developer, “Can*
2415 *Google Cloud Vision generate labels in Spanish via its API? ... [Does the API]*
2416 *allow to select which language to return the labels in?”* [442] points to a lack of
2417 *details relating to the characteristics of the models used by the API. It would seem*
2418 *that developers are unaware of aspects of the technical domain concerns.*

2419 The next most frequent category is **Correctness (A-1a)** with 22.87% of posts. In
2420 the context of the technical domain there are many limits that developers need to be
2421 aware of: range and increments of a model score [81]; required data pre-processing
2422 steps for optimal performance; and features provided by the models (as explained in
2423 Section 5.5.2.2). Considering the relation between technical concerns and software
2424 quality, developers are right to question providers on correctness; “*Are the Cloud*
2425 *Vision API limits in documentation correct?*” [434].

2426 5.6.1.2 *Are the distribution of issues similar to prior studies? (RQ2)*

2427 Visual inspection of Figure 5.3 shows that the distributions for the documentation-
2428 specific classification and the generalised classification are different (compared to
2429 prior studies). As a sanity check we conducted a χ^2 test and calculated the effect
2430 size ϕ_c . We were able to reject the null hypothesis for both classification schemes,
2431 that the distribution of issues were the same as the previous studies (see Section 5.5).
2432 We now discuss the most prominent differences between our study and the previous
2433 studies.

2434 In the context of IWS SO posts, Taxonomy B suggests that Errors (B-3) are
2435 discussed most amongst developers. These results are in contrast to similar studies
2436 made in more *mature* API domains, such as Mobile Development [21, 22, 33, 34, 283]
2437 and Web Development [327]. Here, API Usage (B-1) is much more frequently
2438 discussed, followed by Conceptual (B-5), Discrepancy (B-2) and Errors (B-3). We
2439 argue in the following section that an improved developer understanding can be
2440 achieved by educating them about the IWS lifecycle and the ‘whole’ system that
2441 wraps such services.

²⁴⁴² In the Android study API usage questions (B-1) were the highest category
²⁴⁴³ (28.93% compared to 22.29% in our study). As stated in the analysis of the Error
²⁴⁴⁴ questions this discrepancy could be due to the maturity of the domain. However,
²⁴⁴⁵ another explanation could be the scope of the two individual studies. Beyer et al. [34]
²⁴⁴⁶ used a broad search strategy consisting of posts tagged Android. This search term
²⁴⁴⁷ fetches issues related to the entire Android platform which is significantly larger
²⁴⁴⁸ than searching for computer vision APIs using 229 search terms. As a consequence
²⁴⁴⁹ of more posts and more APIs there would be use cases resulting in additional posts
²⁴⁵⁰ related to API Usage (B-1).

²⁴⁵¹ Applying existing SO taxonomies allowed us to better understand the distribution
²⁴⁵² of the issues across different domains. In particular, the issues raised around IWSs
²⁴⁵³ appear to be primarily due to poor documentation, or insufficient explanation around
²⁴⁵⁴ errors and limitations. Hence, many of the concerns could be addressed by adding
²⁴⁵⁵ more details to the end-point descriptions, and by providing additional information
²⁴⁵⁶ around how these services are designed to work.

²⁴⁵⁷ 5.6.2 The Developer’s Learning Approach

²⁴⁵⁸ In this subsection, we offer an explanation as to why developers are complaining
²⁴⁵⁹ about certain things when trying to use IWSs on SO (RQ1), as characterised through
²⁴⁶⁰ the use of prior SO classification frameworks (RQ2). This is described through
²⁴⁶¹ the theoretical lenses of two learning taxonomies: Bloom’s context complexity and
²⁴⁶² intellectual ability taxonomy, and the Structure of the Observed Learning Outcome
²⁴⁶³ (SOLO) taxonomy (i.e., the nature by which developer’s learn). We argue that the
²⁴⁶⁴ issues with using IWSs relating to the lower-levels of these learning taxonomies
²⁴⁶⁵ are easily solvable by slight fixes and improvements to the documentation of these
²⁴⁶⁶ services. However, the higher dimensions of these taxonomies demand far more
²⁴⁶⁷ rigorous mitigation strategies than documentation alone (potentially more structured
²⁴⁶⁸ education). Thus, many of the questions posted are from developers who are *learning*
²⁴⁶⁹ to *understand* the domain of IWSs and AI, and (hence) both SOLO and Bloom’s
²⁴⁷⁰ taxonomies are applicable for this discussion—as described below within the context
²⁴⁷¹ of our domain—as pedagogical aides.

²⁴⁷² 5.6.2.1 Bloom’s Taxonomy

²⁴⁷³ The cognitive domain under Bloom’s taxonomy [39] consists of six objectives.
²⁴⁷⁴ Within the context of IWSs, developers are likely to ask questions due to causal links
²⁴⁷⁵ that exist in the following layers of Bloom’s taxonomy: (i) *knowledge*, where the
²⁴⁷⁶ developer does not remember or know of the basic concepts of computer vision and
²⁴⁷⁷ AI (in essence, they may think that AI is as smart as a human); (ii) *comprehension*,
²⁴⁷⁸ where the developer does not understand how to interpret basic concepts, or they
²⁴⁷⁹ are mis-understanding how they are used in context; (iii) *application*, where the
²⁴⁸⁰ developer is struggling to apply existing concepts within the context of their own
²⁴⁸¹ situation; (iv) *analysis*, where the developer is unable to analyse the results from IWSs
²⁴⁸² (i.e., understand response objects); (v) *evaluation*, where the developer is unable to
²⁴⁸³ evaluate issues and make use of best-practices when using IWSs; and (vi) *synthesise*,

²⁴⁸⁴ where the developer is posing creative questions to ask if new concepts are possible
²⁴⁸⁵ with CVSSs.

²⁴⁸⁶ **5.6.2.2 SOLO Taxonomy**

²⁴⁸⁷ The SOLO taxonomy [35] consists of five levels of understanding. The causal
²⁴⁸⁸ links behind the SO questions we have found relate to the following layers of the
²⁴⁸⁹ SOLO taxonomy: (i) *pre-structural*, where the developer has a question indicating
²⁴⁹⁰ incompetence or has little understanding of computer vision; (ii) *uni-structural*,
²⁴⁹¹ where the developer is struggling with one key aspect (i.e., a simple question about
²⁴⁹² computer vision); (iii) *multi-structural*, where the developer is questioning multiple
²⁴⁹³ concepts (independently) to understand how to build their system (e.g., system
²⁴⁹⁴ integration with the IWS); (iv) *relational*, where the developer is comparing and
²⁴⁹⁵ contrasting the best ways to achieve something with IWSs; and (v) *extended abstract*,
²⁴⁹⁶ where the developer poses a question theorising, formulating or postulating a new
²⁴⁹⁷ 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.” [438]	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).” [418]	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?” [425]	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?” [432]	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?” [430]	Analysis	Relational

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

²⁴⁹⁹ To understand our findings with the lenses of pedagogical aids, we aligned Tax-
²⁵⁰⁰ onomies A and B to Bloom’s and the SOLO taxonomies for a random sample of 50
²⁵⁰¹ issues described in Section 5.4.3.1. To do this, we reviewed all 50 of these SO posted
²⁵⁰² questions and applied both the Bloom and SOLO taxonomies. The primary author

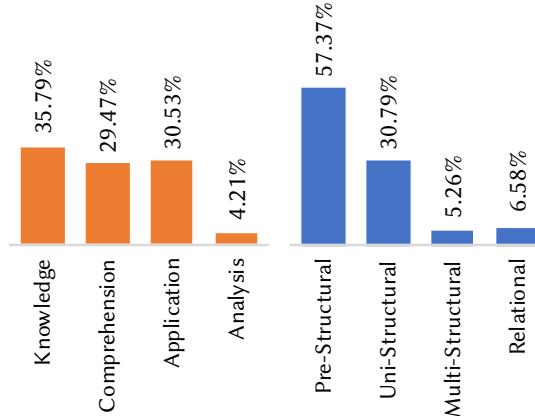


Figure 5.4: Alignment of Bloom (Orange) and SOLO (Blue) taxonomies against Taxonomy A and B dimensions against all 213 classifications made in the random sample of 50 posts.

2503 assigned each of the 50 questions a level within the Bloom and SOLO taxonomies,
 2504 removed out noise (i.e., false positive posts of no relevance to IWSs) and unassigned
 2505 dimensions from reliability agreement, and then compared the relevant dimensions
 2506 of Taxonomy A and B dimensions (not sub-categories). The comparison of align-
 2507 ments of posts to the five SOLO dimensions and six Bloom dimensions are shown
 2508 in Figure 5.4. We acknowledge that this is only an approximation of the current
 2509 state of the developer’s understanding of IWSs. This early model will require further
 2510 studies to perform a more thorough analysis, but we offer this interpretation for early
 2511 discussion.

2512 As shown in Figure 5.4, the bulk of the posts fall in the lower constructs of
 2513 Bloom’s and the SOLO taxonomy. This indicates that modification to certain doc-
 2514 umentation aspects can address many of these issues. For example, many issues
 2515 can be ratified with better descriptions of response data and error messages: “*I was*
 2516 *exploring google vision and in the specific function ‘detectCrops’, gives me the crop*
 2517 *hints. what does this means exactly?”* [433]; “*I am making a very simple API call*
 2518 *to the Google Vision API, but all the time it’s giving me error that ‘google.oauth2’*
 2519 *module not found.”* [448]

2520 However, and more importantly, the higher-construct questions ranging from
 2521 the middle of the third dimensions on are not as easily solvable through improved
 2522 documentation (i.e., apply and multi-structural) which leaves 34.74% (Bloom’s)
 2523 and 11.84% (SOLO) unaccounted for, resolvable only through improved education
 2524 practices.

2525 5.6.3 Implications

2526 5.6.3.1 For Researchers

2527 **Investigate the evolution of post classification** Analysing how the distribution of
 2528 the reported issues changes over time would be an important study. This study could

2529 answer questions such as ‘*Does the evolution of IWSs follow the same pattern as*
2530 *previous software engineering trends such as mobile app or web development?*’ As
2531 with any new emerging field, it is key to analyse how developers perceive such issues
2532 over time. For instance, early issues with web or mobile app development matured
2533 as their respective domain matured, and we would expect similar results to occur
2534 in the IWSs space. Future researchers could plan for a longitudinal study, such as
2535 a long-term survey with developers to gather their insights in this evolving domain,
2536 reviewing case studies of projects that use intelligent web services from now into
2537 the future, or re-mining SO at a later date and comparing the results to this study.
2538 This will help assess evolving trends and characteristics, and determine how and if
2539 the nature of the developer’s experience with IWSs (and AI in general) changes with
2540 time.

2541 **Investigate the impact of technical challenges on API usage** As discussed above,
2542 IWSs have characteristics that may influence API usage patterns and should be
2543 investigated as a further avenue of research. Further mining of open source software
2544 repositories that make use of IWSs could be assessed, thereby investigating if API
2545 patterns evolve with the rise of AI-based applications.

2546 *5.6.3.2 For Educators*

2547 **Education on high-level aspects of IWSs** As demonstrated in our analysis of their
2548 SO posts, many developers appear to be unaware of the higher-level concepts that
2549 exist within the AI and ML realm. This includes the need to pre- and post-process
2550 data, the data dependency and instability that exists in these services, and the specific
2551 algorithms that empower the underlying intelligence and hence their limitations and
2552 characteristics. However, most developers don’t seem to complain about these factors
2553 due to the lack of documentation (i.e., via Taxonomy A). Rather, they are unaware
2554 that such information should be documentation and instead ask generalised and open
2555 questions (i.e., via Taxonomy B). Thus, documentation improvements alone may not
2556 be enough to solve these issues. This results in uncertainty during the preparation
2557 and operation (usage) of such services. Such high-level conceptual information is
2558 currently largely missing in developer documentation for IWSs. Furthermore, many
2559 of the background ML and AI algorithm information needed to understand and use
2560 intelligent systems in context are built within data science (not software engineering)
2561 communities. A possible road-map to mitigate this issue would be the development
2562 of a software engineer’s ‘crash-course’ in ML and AI. The aim of such a course
2563 would encourage software engineers to develop an appreciation of the nuances and
2564 the inherent risks and implications that comes with using IWSs. This could be
2565 taught at an undergraduate level to prepare the next generation of developers of a
2566 ‘programming 2.0’ era. However, the key aspects and implications that are presented
2567 with AI would need to be well-understood before such a course is developed, and
2568 determining the best strategy to curate the content to developers would be best left
2569 to the software engineering education domain. Further investigation in applying
2570 educational taxonomies in the area (such as our attempts to interpret our findings

2571 using Bloom's and the SOLO taxonomies) would need to be thoroughly explored
2572 beforehand.

2573 *5.6.3.3 For Software Engineers*

2574 **Better understanding of intelligent API contextual usage** Our results show that
2575 developers are still learning to use these APIs. We applied two learning perspectives
2576 to interpret our results. In applying the two pedagogical taxonomies to our findings,
2577 we see that most issues seem to fall into the pre-structural and knowledge-based
2578 categories; little is asked of higher level concepts and a majority of issues do not
2579 offer complex analysis from developers. This suggests that developers are struggling
2580 as they are unaware of the vocabulary needed to actually use such APIs, further
2581 reinforcing the need for API providers to write overview documentation (as noted in
2582 prior work [80]) and not just simple endpoint documentation. This said, improved
2583 documentation isn't always enough—as suggested by our discussion in Section 5.6.2,
2584 software engineers should explore further education to attain a greater appreciation
2585 of the nuances of ML when attempting to use these services.

2586 *5.6.3.4 For Intelligent Service Providers*

2587 **Clarify use cases for IWSs** Inspecting SO posts revealed that there is a level of
2588 confusion around the capabilities of different IWSs. This needs to be clarified in
2589 associated API documentation. The complication with this comes with targeting
2590 the documentation such that software developers (who are untrained in the nuances
2591 of AI and ML as per Section 5.6.3.2) can to digest it and apply it in-context to
2592 application development.

2593 **Technical domain matters** More needs to be provided than a simple endpoint
2594 description as conventional APIs offer by describing the whole framework by which
2595 the endpoint sits, giving further context. This said, compared to traditional APIs,
2596 we find that developers complain less about the documentation and more about
2597 shallower issues. All expected pre-processing and post-processing needs to be
2598 clearly explained. A possible mitigation to this could be an interactive tutorial that
2599 helps developers fully understand the technical domain using a hands-on approach.
2600 For example, websites offer interactive Git tutorials⁷ to help developers understand
2601 and explore the technical domain matters under version control in their own pace.

2602 **Clarify limitations** API developers need to add clear limitations of the existing
2603 APIs. Limitations include list of objects that can be returned from an endpoint. We
2604 found that the cognitive anchors of how existing, conventional API documentation
2605 is written has become ‘ported’ to the computer vision realm, however a lot more
2606 overview documentation than what is given at present (i.e., better descriptions of
2607 errors, improved context of how these systems work in etc.) needs to be given. Such
2608 documentation could be provided using interactive tutorials.

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

2609 5.7 Threats to Validity**2610 5.7.1 Internal Validity**

2611 As detailed in Section 5.4.3.1, Taxonomies A and B present slight and fair agreement,
2612 respectively, when inter-rater reliability was applied. The nature of our disagree-
2613 ments largely fell due to the subjectivity in applying either taxonomies to posts.
2614 Despite all coders agreeing to the shared interpretation of both taxonomies, both
2615 taxonomies are subjective in their application, which was not reported by either
2616 Aghajani et al. or Beyer et al.. In many cases, multi-label classification seemed ap-
2617 propriate, however both taxonomies use single-label mapping which we find results
2618 in too much subjectivity. This subjectivity, therefore, ultimately adversely affects
2619 inter-rater reliability (IRR) analysis. Thus, a future mitigation strategy for similar
2620 work should explore multi-label classification to avoid this issue; Beyer et al., for
2621 example, plan for multi-label classification as future work. However, these studies
2622 would need to consider the statistical challenges in calculating multi-rater, multi-
2623 label IRR for thorough reliability analysis in addressing subjectivity. The selection
2624 of SO posts used for our labelling, chiefly in the subjectivity of our classifications, is
2625 of concern. We mitigate this by an extensive review process assessing the reliability
2626 of our results as per Section 5.4.3.1. The classification of our posts into the SOLO
2627 and Bloom’s taxonomies was performed by the primary author only, and therefore
2628 no inter-rater reliability statistics were performed. However, we used these peda-
2629 gogy related taxonomies as a lens to gain an additional perspective to interpret our
2630 results. Future studies should attempt a more rigorous analysis of SO posts using
2631 Bloom’s and SOLO taxonomies. We only aligned posts to one category for each
2632 taxonomy and did not align these using multi-label classification. This brings more
2633 complexity to the analysis, and our attempts to repeat prior studies’ methodologies
2634 (see Section 5.3). Multi-label classification for IWSs SO posts is an avenue for future
2635 research.

2636 5.7.2 External Validity

2637 While every effort was made to select posts from SO relevant to CVSs, there are
2638 some cases where we may have missed some posts. This is especially due to the
2639 case where some developers mis-reference certain IWSs under different names (see
2640 Section 5.4.2.1).

2641 Our SOLO and Bloom’s taxonomy analysis has only been investigated through
2642 the lenses of IWSs, and not in terms of conventional APIs (e.g., Andriod APIs).
2643 Therefore, we are not fully certain how these results found would compare to other
2644 types of APIs. Two *existing* SO classification taxonomies were used rather than
2645 developing our own. We wanted to see if previous SO taxonomies could be applied
2646 to IWSs before developing a new, specific taxonomy, and these taxonomies were
2647 applied based on our interpretation (see Section 5.4.2.4) and may not necessarily
2648 reflect the interpretation of the original authors. Moreover, automated techniques
2649 such as topic modelling were not utilised as we found these produce descriptive
2650 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

2683

2684

2685

Ranking Computer Vision Service Issues using Emotion[†]

2686

2687 **Abstract** Software developers are increasingly using intelligent web services to implement
2688 ‘intelligent’ features. Studies show that incorporating artificial intelligence (AI) into an
2689 application increases technical debt, creates data dependencies, and introduces uncertainty
2690 due to non-deterministic behaviour. However, we know very little about the emotional state
2691 of software developers who deal with such issues. In this paper, we do a landscape analysis
2692 of emotion found in 1,425 Stack Overflow (SO) posts about computer vision services. We
2693 investigate the application of an existing emotion classifier EmoTxt and manually verify our
2694 results. We found that the emotion profile varies for different question categories.

2695 6.1 Introduction

2696 Recent advances in artificial intelligence have provided software engineers with
2697 new opportunities to incorporate complex machine learning capabilities, such as
2698 computer vision, through cloud-based intelligent web services (IWSs). These new
2699 set of services, typically offered as API calls are marketed as a way to reduce the
2700 complexity involved in integrating AI-components. However, recent work shows that
2701 software engineers struggle to use these IWSs [84]. Furthermore, the accompanying
2702 documentation fails to address common issues experienced by software engineers
2703 and, often, engineers resort to online communication channels, such as, JIRA and
2704 Stack Overflow (SO) to seek advice from their peers [84].

2705 While seeking advice on the issues, software engineers tend to express their emotions
2706 (such as frustration or confusion) within the questions. Recognising the value
2707 of considering emotions, other researchers have investigated emotions expressed by
2708 software developers within communication channels [249] including SO [62, 242];

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

2709 the broad motivation of these works is to generally understand the emotional land-
2710 scape and improve developer productivity [118, 230, 249]. However, previous works
2711 have not directly focused on the nature of emotions expressed in questions related to
2712 IWSs. We also do not know if certain types of questions express stronger emotions.

2713 The machine-learnt behaviour of these IWSs is typically non-deterministic and,
2714 given the dimensions of data used, their internal inference process is hard to reason
2715 about [81]. Compounding the issue, documentation of these cloud systems does not
2716 explain the limits, nor how they were created (esp. data sets used to train them).
2717 This lack of transparency makes it difficult for even senior developers to properly
2718 reason about these systems, so their prior experience and anchors do not offer
2719 sufficient support [84]. In addition, adding machine learned behaviour to a system
2720 incurs ongoing maintenance concerns [295]. There is a need to better understand
2721 emotions expressed by developers to inform cloud vendors and help them improve
2722 their documentation and error messages provided by their services.

2723 This work builds on top of recent work that explored *what* pain-points developers
2724 face when using IWSs through a general analysis of 1,425 SO posts (questions) [84]
2725 using an existing SO issue classification taxonomy [34]. In this work, we consider
2726 the emotional state expressed within these pain-points, using the same data set of
2727 1,425 SO posts. We identify the emotions in each SO question, and investigate if
2728 the distribution of these emotions is similar across the various types of questions.

2729 In order to classify emotions from SO posts, we use EmoTxt, a recently proposed
2730 toolkit for emotion recognition from text [61, 62, 242]. EmoTxt has been trained
2731 and built on SO posts using the emotion classification model proposed by Shaver
2732 et al. [300]. The category of issue was manually determined in our prior work.

2733 The key findings of our study are:

- 2734 • The distribution of emotions is different across the taxonomy of issues.
- 2735 • A deeper analysis of the results, obtained from the EmoTxt classifier, suggests
2736 that the classification model needs further refinement. Love and joy, the
2737 least expected emotions when discussing API issues, are visible across all
2738 categories.
- 2739 • A different emotion classification scheme is required to better reflect the
2740 emotions within the questions.

2741 In order to promote future research and permit replication, we make our data
2742 set publicly available.¹ The paper structure is as follows: Section 6.2 provides
2743 an overview on prior work surrounding the classification of emotions from text;
2744 Section 6.3 describes our research methodology; Section 6.4 presents the results
2745 from the EmoTxt classifier; Section 6.5 provides a discussion of the results obtained;
2746 Section 6.6 outlines the threats to validity; Section 6.7 presents the concluding
2747 remarks.

¹See <http://bit.ly/2RiULgW>.

2748 6.2 Emotion Mining from Text

2749 Several studies have investigated the role of emotions generally in software development [118, 249, 301, 352]. Work in the area of behavioural software engineering
2750 established the link between software developer's happiness and productivity [133].
2751 Wrobel [352] investigated the impact that software developers' emotion has on the
2752 development process and found that frustration and anger were amongst the emotions
2753 that posed the highest risk to developer's productivity.
2754

2755 Recent studies focused on emotion mining from text within communication chan-
2756 nels used by software engineers to communicate with their peers [118, 230, 242,
2757 249]. Murgia et al. [230] and Ortú et al. [249] investigated the emotions expressed
2758 by developers within an issue tracking system, such as JIRA, by labelling issue com-
2759 ments and sentences written by developers using Parrott's framework. Gachechiladze
2760 et al. [118] applied the Shaver framework to detect anger expressed in comments
2761 written by developers in JIRA. The Collab team [61, 242] extended the work done
2762 by Ortú et al. [249] and developed a gold standard data set collected from SO
2763 posts consisting of questions, comments and feedback. This data set was manually
2764 annotated using the Shaver's emotion model. The Shaver's model consists of a tree-
2765 structured, three level, hierarchical classification of emotions. The top level consists
2766 of six basic emotions namely, love, joy, anger, sadness, fear and surprise [300]. The
2767 subsequent levels further refines the granularity of the previous level. One of their
2768 recent work [242] involved 12 raters to manually annotate 4,800 posts (where each
2769 post included the question, answer and comments) from SO. The same question
2770 was assigned to three raters to reduce bias and subjectivity. Each coder was re-
2771 quested to indicate the presence/absence of each of the six basic emotions from the
2772 Shaver framework. As part of their work they developed an emotion mining toolkit,
2773 EmoTxt [61]. The work conducted by the Collab team is most relevant to our study
2774 since their focus is on identifying emotion from SO posts and their toolkit is trained
2775 on a large data set of SO posts.

2776 6.3 Methodology

2777 As mentioned in our introduction, this paper uses the data set reported in Cummaudo
2778 et al.'s ICSE 2020 paper [84]. As this paper is in press, we reproduce a summary
2779 of the methodology used in constructing this data set methodology below. For full
2780 details, we refer to the original paper. Supplementary materials used for this work
2781 are provided for replication.¹

2782 Our research methodology consisted of the following steps: (i) data extraction
2783 from SO resulting in 1,425 questions about intelligent computer vision services
2784 (CVSs); (ii) question classification using the taxonomy presented by Beyer et al. [34];
2785 (iii) automatic emotion classification using EmoTxt based on Shaver et al.'s emotion
2786 taxonomy [300]; and (iv) manual classification of 25 posts to better understand
2787 developers emotion. We calculated the inter-rater reliability between EmoTxt and
2788 our manually classified questions in two ways: (i) to see the overall agreement
2789 between the three raters in applying the Shaver et al. emotions taxonomy, and (ii) to

²⁷⁹⁰ see the overall agreement with EmoTxt’s classifications. Further details are provided
²⁷⁹¹ below.

²⁷⁹² 6.3.1 Data Set Extraction from Stack Overflow

²⁷⁹³ 6.3.1.1 Intelligent Service Selection

²⁷⁹⁴ We contextualise this work within popular CVS providers: Google Cloud [388],
²⁷⁹⁵ AWS [363], Azure [402] and IBM Cloud [398]. We chose these four providers given
²⁷⁹⁶ their prominence and ubiquity as cloud service vendors, especially in enterprise
²⁷⁹⁷ applications [277]. We acknowledge other services beyond the four analysed which
²⁷⁹⁸ provide similar capabilities [376, 377, 384, 397, 449, 450]. Additionally, only
²⁷⁹⁹ English-speaking services have been selected, excluding popular CVSs from Asia
²⁸⁰⁰ (e.g., [374, 375, 396, 415, 416]).

²⁸⁰¹ 6.3.1.2 Developing a search query

²⁸⁰² To understand the various ways developers refer to these services, we needed to find
²⁸⁰³ search terms that are commonplace in question titles and bodies that discuss the
²⁸⁰⁴ service names. One approach is to use the *Tags* feature in SO. To discover which
²⁸⁰⁵ tags may be relevant, we ran a search² within SO against the various brand names of
²⁸⁰⁶ these CVSs, reviewed the first three result pages, and recorded each tag assigned per
²⁸⁰⁷ question.³ However, searching using tags alone on SO is ineffective (see [23, 320]).
²⁸⁰⁸ To overcome this limitation, we ran a second query within the Stack Exchange Data
²⁸⁰⁹ Explorer⁴ (SEDE) using these tags, we sampled 100 questions (per service), and
²⁸¹⁰ noted the permutations in how developers refer to each service⁵. We noted 229
²⁸¹¹ permutations.

²⁸¹² 6.3.1.3 Executing our search query

²⁸¹³ Next, we needed to extract questions that make reference to any of these 229 per-
²⁸¹⁴ mutations. SEDE has a 50,000 row limit and does not support case-insensitivity,
²⁸¹⁵ however Google’s BigQuery does not. Therefore, we queried Google’s SO dataset
²⁸¹⁶ on each of the 229 terms that may occur within the title or body of question posts,⁶
²⁸¹⁷ which resulted in 21,226 questions.

²⁸¹⁸ 6.3.1.4 Refining our inclusion/exclusion criteria

²⁸¹⁹ To assess the suitability of these questions, we filtered the 50 most recent posts
²⁸²⁰ as sorted by their *CreationDate* values. This helped further refine the inclusion
²⁸²¹ 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

²⁸²² ‘GCV’, ‘WCS’⁷) allowed for false positive questions to be included, which were
²⁸²³ removed. Furthermore, we consolidated all overlapping terms (e.g., ‘Google Vision
²⁸²⁴ API’ was collapsed into ‘Google Vision’) to enhance the query. Additionally, we
²⁸²⁵ reduced our 221 search terms to just 27 search terms by focusing on CVSs *only*⁸
²⁸²⁶ which resulted in 1,425 questions. No duplicates were recorded as determined by
²⁸²⁷ the unique ID, title and timestamp of each question.

²⁸²⁸ 6.3.1.5 *Manual filtering*

²⁸²⁹ The next step was to assess the suitability and nature of the 1,425 questions extracted.
²⁸³⁰ The second author ran a manual check on a random sample of 50 posts, which were
²⁸³¹ parsed through a templating engine script⁹ in which the ID, title, body, tags, created
²⁸³² date, and view, answer and comment counts were rendered for each post. Any match
²⁸³³ against the 27 search terms in the title or body of the post were highlighted, in which
²⁸³⁴ three false positives were identified as either library imports or stack traces, such
²⁸³⁵ as `aws-java-sdk-rekognition.jar`. In addition, we noted that there were false
²⁸³⁶ positive hits related to non-CVSs. We flagged posts of such nature as ‘noise’ and
²⁸³⁷ 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>.

2838 6.3.2 Question Type & Emotion Classification

2839 6.3.2.1 Manual classification of question category

2840 We classify our 1,425 posts using Beyer et al.'s taxonomy [34] as it was comprehensive and validated [84]. We split the posts into 4 additional random samples, in
 2841 addition to the random sample of 50 above. 475 posts were classified by the second
 2842 author and three other research assistants¹⁰ classified the remaining 900 (i.e., a total
 2843 of 1,375 classifications). An additional 450 classifications were assigned due to
 2844 reliability analysis, in which the remaining 50 posts were classified nine times by
 2845 various researchers in our group.¹¹

2846 Due to the nature of reliability analysis, multiple classifications (450) existed
 2847 for these 50 posts. Therefore, we applied a 'majority rule' technique to each post
 2848 allowing for a single classification assignment and therefore analysis within our re-
 2849 sults. When there was a majority then we used the majority classification; when
 2850 there was a tie, then we used the classification that was assigned the most out of the
 2851 entire 450 classifications. As an example, 3 raters classified a post as *API Usage*,
 2852 1 rater classified the same post as a *Review* question and 5 raters classified the post
 2853 as *Conceptual*, resulting in the post being classified as a *Conceptual* question. For
 2854 another post, three raters assigned *API Usage*, *Discrepancy* and *Learning* (respec-
 2855 tively), while 3 raters assigned *Review* and 3 raters assigned *Conceptual*. In this
 2856 case, *Review* and *Conceptual* were tied, but was resolved down to *Conceptual* as this
 2857 classification received 147 more votes than *Review* across all classifications made in
 2858 the sample of 50 posts.

2859 However, where a post was extracted from our original 1,425 posts but was either
 2860 a false positive, not applicable to IWSs (see Section 6.3.1.5), or not applicable to
 2861 a taxonomy dimension/category, then the post was flagged for removal in further
 2862 analysis. This was done 180 times, leaving a total of 1,245 posts.

2863 Our interpretation Beyer et al.'s taxonomy is provided in Table 6.1, which
 2864 presents a transcription of *our understanding* of the respective taxonomy. We
 2865 baselined all coding against *our interpretation only*, and thus our classifications
 2866 are therefore independent of Beyer et al.'s findings, since we baseline results via
 2867 Table 6.1's interpretation.

2868 6.3.2.2 Emotion classification using AI techniques

2869 After extracting and classifying all posts, we then piped in the body of each question
 2870 into a script developed to remove all HTML tags, code snippets, blockquotes and
 2871 hyperlinks, as suggested by Novielli et al. [242]. We replicated and extended the
 2872 study conducted by Novielli et al. [242] on our data set derived from 1,425 SO posts,
 2873 consisting of questions only. Our study consisted of three main steps, namely, (1)
 2874 automatic emotion classification using EmoTxt, (2) manual annotation process and,
 2875 (3) comparison of the automatic classification result with the manually annotated
 2876 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 [84].

2878 6.3.2.3 *Emotion classification using EmoTxt*

2879 We started with a file containing 1,245 non-noise SO questions, each with an as-
2880 sociated question type as classified using the strategy discussed in Section 6.3.2.1.
2881 We pre-processed this file by extracting the question ID and body text to meet the
2882 format requirements of the EmoTxt classifier [61]. This classifier was used as it
2883 was trained on SO posts as discussed in Section 6.2. We ran the classifier for each
2884 emotion as this was required by EmoTxt model. This resulted in 6 output prediction
2885 files (one file for each emotion: *Love*, *Joy*, *Surprise*, *Sadness*, *Fear*, *Anger*). Each
2886 question within these files referenced the question ID and a predicted classification
2887 (YES or NO) of the emotion. We then merged the emotion prediction files into an
2888 aggregate file with question text and Beyer et al.’s taxonomy classifications. This
2889 resulted in 796 emotion classifications. We further analysed the classifications and
2890 generated an additional classification of *No Emotion* for the 622 questions where
2891 EmoTxt predicted NO for all the emotion classification runs.

2892 Of the 796 questions with emotion detected, 143 questions had 2 or more
2893 emotions predicted: 1 question¹² had up to 4 emotions detected (*Surprise*, *Sadness*,
2894 *Joy* and *Fear*), 28 questions had up to 3 emotions detected, and the remaining 114
2895 had up to two emotions detected.

2896 6.3.2.4 *Manual Annotation Process*

2897 In order to evaluate and also better understand the process used by EmoTxt to
2898 classify emotions, we manually annotated a small sample of 25 SO posts, randomly
2899 selected from our data set. Each of these 25 posts were assigned to three raters who
2900 carried out the following three steps: (i) identify the presence of an emotion; (ii)
2901 if an emotion(s) exists, classify the emotion(s) under one of the six basic emotions
2902 proposed by the Shaver framework [300]; (iii) if no emotion is identified, annotate as
2903 neutral. We then collated all rater’s results and calculated Light’s Kappa (L_K) [197]
2904 to measure the overall agreement *between* raters to measure the similarity in which
2905 independent raters classify emotions to SO posts. As L_K does not support multi-class
2906 classification (i.e., multiple emotions) per subjects (i.e., per SO post), we binarised
2907 the results each emotion and rater as TRUE or FALSE to indicate presence, calculated
2908 the L_K per emotion against the three raters, and averaged the result across all emotions
2909 to get an overall strength of agreement.

2910 6.3.2.5 *Comparing EmoTxt results with the results from Manual Classification*

2911 The next step involved comparing the ratings of the 25 SO posts that were manually
2912 annotated by the three raters with the results obtained for the same set of 25 SO
2913 posts from the EmoTxt classifier. Similar to Section 6.3.2.4, we used Cohen’s Kappa
2914 (C_K) [75] to measure the consistency of classifications of EmoTxt’s classifications
2915 versus the manual classifications of each rater. We separated the classifications
2916 per emotion and calculated C_K for each rater against EmoTxt and averaged these
2917 values for all emotions. After noticing poor results, the three raters involved in

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

²⁹¹⁸ Section 6.3.2.4 were asked to compare and discuss the ratings from the EmoTxt
²⁹¹⁹ classifier against the manual ratings.

²⁹²⁰ The findings from this process are presented and discussed in the next two
²⁹²¹ sections.

²⁹²² 6.4 Findings

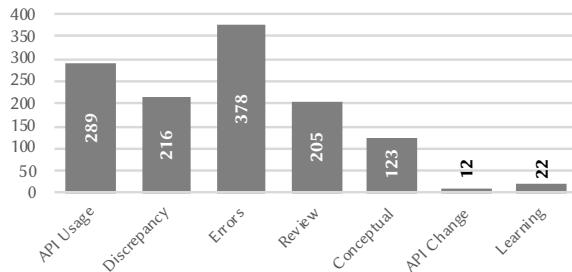


Figure 6.1: Distribution of SO question types.

²⁹²³ Figure 6.1 displays the overall distribution of question types from the 1,245
²⁹²⁴ posts classified in [84], when adjusted for majority ruling as per Section 6.3.2.1. It
²⁹²⁵ is evident that developers ask issues predominantly related to API errors when using
²⁹²⁶ CVSs and, additionally, how they can use the API to implement specific functionality.
²⁹²⁷ 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

²⁹²⁸ Table 6.2 displays the frequency of questions that were classified by EmoTxt
²⁹²⁹ when compared to our assignment of question types, while Figure 6.2 presents the
²⁹³⁰ emotion data proportionally across each type of question. *No Emotion* was the
²⁹³¹ most prevalent across all question types, which is consistent with the findings of the
²⁹³² Collab group during the training of the EmoTxt classifier. Interestingly, *API Change*
²⁹³³ questions had a distinct distribution of emotions, where 31.25% of questions had *No*
²⁹³⁴ *Emotion* compared to the average of 42.01%. This is likely due to the low sample
²⁹³⁵ size of *API Change* questions, with only 12 assignments, however the next highest
²⁹³⁶ set of emotive questions are found in the second largest sample (*API Usage*, at
²⁹³⁷ 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 [192], 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.

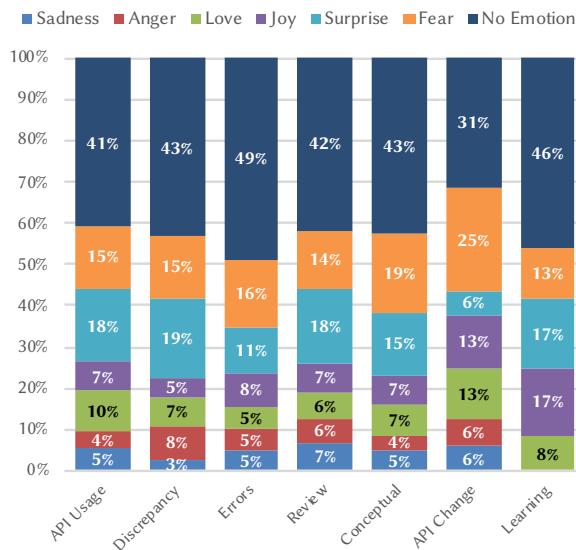


Figure 6.2: Proportion of emotions per question type.

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

2961 results from our reliability analysis, an introspection of the data set sheds some light
2962 to the discrepancy. For example, question 55113529 shows no indication of *Joy*,
2963 rather the developer is expressing a state of confusion. The phrase “*Thanks for your*
2964 *help*” could be the reason why the miss-classification occurred if words like “thanks”
2965 were associated with joy. However, in this case, it seems unlikely that the developer
2966 is expressing joy as the developer has followed a tutorial but is still encountering
2967 an error. Similarly, question 39797164, classified as *Love* and question 50500341,
2968 classified as *Sadness* express a state of confusion and the urge to know more about the
2969 product; upon inspecting the entire question in context, it is difficult to consistently
2970 agree with the emotions as determined by EmoTxt, and further exploration into the
2971 behaviour and limitations of the model is necessary.

2972 Our results indicate further work is needed to refine the machine learning (ML)
2973 classifiers that mine emotions in the SO context. The question that arises is whether
2974 the classification model is truly reflective of real-world emotions expressed by soft-
2975 ware developers. As highlighted by Curumsing [87], the divergence of opinions with
2976 regards to the emotion classification model proposed by theorists raises doubts to
2977 the foundations of basic emotions. Most of the studies conducted in the area of emotion
2978 mining from text is based on an existing general purpose emotion framework
2979 from psychology [57, 242, 249]—none of which are tuned for software engineering
2980 domain. In our our study, we note the emotions expressed by software develop-
2981 ers within SO posts are quite narrow and specific. In particular, emotions such as
2982 frustration and confusion would be more appropriate over love and joy.

2983 6.6 Threats to Validity

2984 6.6.1 Internal Validity

2985 The *API Change* and *Learning* question types were few in sample size (only 12 and
2986 22 questions, respectively). The emotion proportion distribution of these question
2987 types are quite different to the others. Given the low number of questions, the sample
2988 is too small to make confident assessments. Furthermore, our assignment of Beyer
2989 et al.’s question type taxonomy was single-label; a multi-labelled approach may work
2990 better, however analysis of results would become more complex. A multi-labelled
2991 approach would be indicative for future work.

2992 6.6.2 External Validity

2993 EmoTxt was trained on questions, answers and comments, however our data set
2994 contained questions only. It is likely that our results may differ if we included other
2995 discussion items, however we wished to understand the emotion within developers’
2996 *questions* and classify the question based on the question classification framework
2997 by Beyer et al. [34]. Moreover, this study has only assessed frustrations within the
2998 context of a concrete domain of CVSs. The generalisability of this study to other
2999 IWSs, such as natural language processing services, or conventional web services,
3000 may be different. Furthermore, we only assessed four popular CVSs; expanding the

3001 data set to include more services, including non-English ones, would be insightful.
3002 We leave this to future work.

3003 **6.6.3 Construct Validity**

3004 Some posts extracted from SO were false positives. Whilst flagged for removal
3005 (Section 6.3.1.5), we cannot guarantee that all false positives were removed. Fur-
3006 thermore, SO is known to have questions that are either poorly worded or poorly
3007 detailed, and developers sometimes ask questions without doing any preliminary
3008 investigation. This often results in down-voted questions. We did not remove such
3009 questions from our data set, which may influence the measurement of our results.

3010 **6.7 Conclusions**

3011 In this paper we analysed SO posts for emotions using an automated tool and cross-
3012 checked it manually. We found that the distribution of emotion differs across the
3013 taxonomy of issues, and that the current emotion model typically used in recent
3014 works is not appropriate for emotions expressed within SO questions. Consistent
3015 with prior work [199], our results demonstrate that machine learning classifiers for
3016 emotion are insufficient; human assessment is required.

CHAPTER 7

3017

3018

3019

Better Documenting Computer Vision Services[†]

3020

3021 **Abstract** Using cloud-based computer vision services (CVSs) is gaining traction with
3022 developers for many applications for many reasons: developers can simply access these
3023 AI-components through familiar RESTful APIs, and need not orchestrate large training and
3024 inference infrastructures or curate and label large training datasets. However, while their
3025 APIs *seem* familiar to use, their non-deterministic run-time behaviour and evolution profile
3026 are not adequately communicated to developers, and this results in developers struggling
3027 to use such APIs in-practice. Therefore, improving these services' API documentation is
3028 paramount, as a more complete document facilities the development process of intelligent
3029 software. This study presents an analysis of what facets a 'complete' API document should
3030 have, as synthesised into a taxonomy from 21 academic studies via a systematic mapping
3031 study. We triangulate these findings from literature against 83 developers to assess the
3032 efficacy and utility in-practice of such knowledge. We produce two weighted 'scores'
3033 for each dimension in our taxonomy based on (i) the number of papers producing these
3034 outcomes and their citation count and (ii) the extent to which developers *agree* with the
3035 recommendations arising from these studies (based on our survey). Furthermore, we apply
3036 the taxonomy to three popular CVSs and assess their compliance, producing a third 'score'
3037 using the taxonomy to identify 12 suggested improvements to the API documentation of
3038 these intelligent web services.

3039 7.1 Introduction

3040 Improving API documentation quality is a valuable task for any API—an extensive
3041 API document facilitates productivity, and therefore improved quality is better
3042 engineered into a system [220]. Where application developers integrate new services
3043 (such as computer vision services (CVSs) [81]) 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.

3044 productivity is affected either by inadequate skills (“*I’ve never used an API like*
3045 *this, so must learn from scratch*”) or, where their skills are adequate, an imbalanced
3046 cognitive load that causes excessive context switching (“*I have the skills for this, but*
3047 *am confused or misunderstand*”). This is commonly seen in the emerging computer
3048 vision web services space, where the documentation does not yet completely or
3049 correctly describe the APIs in full [84].

3050 What causes a developer to be confused and how to mitigate it via an improved
3051 API document has been largely explored for conventional APIs. Various studies
3052 have provided a myriad of recommendations based on both qualitative and quantita-
3053 tive analysis of developer opinion. Such recommendations propose ways by which
3054 developers, managers and solution architects can construct systems better with im-
3055 proved documentation. However, while previous works have covered certain aspects
3056 of API usage, many have lacked a systematic review of literature and do not offer a
3057 taxonomy to consolidate these guidelines together. For example, some studies have
3058 considered the technical implementation improving API usability or tools to gener-
3059 ate (or validate) API documentation from its source code (e.g., [208, 243, 341]); still
3060 lacks a consolidated effort to capture recommendations on how to *manually write*
3061 complete, correct, and effective API documentation. The works that *do* produce
3062 these recommendations from literature are largely scattered across multiple sources,
3063 and systematically capturing the information into a readily accessible, consolidated
3064 framework (designed to assist writing API documentation) must be validated in
3065 real-world circumstances to assess its efficacy with practitioners and existing docu-
3066 mentation [80].

3067 As a real-world use case, consider an intelligent web service (IWS)—such as
3068 CVSs—in which an AI-based component produces a non-deterministic result based
3069 on a machine-learnt data-driven algorithm, rather than a predictable, rule-driven
3070 one [81]. These services use machine intelligence to make predictions on images
3071 such as object labelling or facial recognition [363, 374, 375, 376, 377, 384, 388,
3072 396, 397, 398, 402, 415, 416, 449, 450]. The impacts of poor and incomplete
3073 documentation results in developer complaints on online discussion forums such as
3074 Stack Overflow [84]. Many comments show that developers do not think in the
3075 non-deterministic mental model of the designers who created the CVSs. They ask
3076 many varied questions from their peers to try and clarify their understanding.

3077 This paper significantly extends our previous work [80] by evaluating our API
3078 documentation taxonomy in two additional contexts. In our previous work, we
3079 developed a weighted metric for each dimension and category based on how many
3080 literary sources agree that the aspects of our taxonomy should be implemented.
3081 We refer to this as an ‘in-literature’ agreement score. We build upon this facet
3082 but *in-practice* by assessing the efficacy of our taxonomy against developers using
3083 a survey built upon an interpretation of the System Usability Scale (SUS) [55].
3084 We produce a second weighting for the dimensions and categories of the taxonomy,
3085 referred to as a ‘*in-practice*’ agreement score. We then compare both the in-literature
3086 and *in-practice* scores directly, thereby contrasting the statistical agreement the two
3087 have. Lastly, we assess the taxonomy against three popular CVSs, namely Google
3088 Cloud Vision [388], Amazon Rekognition [363] and Azure Computer Vision [402].

3089 For each category in our taxonomy, we assess whether the respective service's
3090 documentation contains, partially-contains or does not contain the recommendation.
3091 From this, we triangulate each category's in-literature and in-practice score against
3092 the service's level of inclusion of the recommendation, thereby making a judgement
3093 as to where the services can improve their documentation to make them more
3094 complete.

3095 The primary contributions in this work are:

- 3096 • a systematic mapping study (SMS) consisting of 21 studies that capture what
3097 knowledge or artefacts should be contained within API documentation;
- 3098 • a five dimensional taxonomy consisting of 34 recommendations based on those
3099 consolidated from the 21 studies;
- 3100 • a score metric for each recommendation based on the number of papers that
3101 agree with the recommendation;
- 3102 • a score metric assessing the efficacy of the 34 recommendations that empirically
3103 reflects what is important to document from a *practitioner* point of view;
3104 and,
- 3105 • a heuristic validation of each recommendation against CVSs, assessing where
3106 existing CVS API documentation needs improvement.

3107 After performing our SMS on what API knowledge should be captured in documentation
3108 to assist API designers, we propose our taxonomy consisting of the
3109 following dimensions: (1) Usage Description; (2) Design Rationale; (3) Domain
3110 Concepts; (4) Support Artefacts; and (5) Documentation Presentation. Following
3111 this, we adopted the SUS surveying technique to assess the overall utility of each
3112 of these recommendations, producing a survey consisting of 43 questions. This
3113 survey was then tested three times within our research group: firstly against three
3114 researchers for feedback on the survey's design, secondly against three software
3115 engineers in our research group with varying levels of experience for developers
3116 for test-retest reliability [179], thirdly against 22 software engineers in our research
3117 group for wider feedback on the survey. Given these feedback improvements, we
3118 surveyed 83 external developers between May 2019 to October 2019, and then analysed
3119 the relevance of each recommendation from the practitioner's viewpoint. We
3120 also assessed the three CVSs for inclusion of each recommendation, and once our
3121 surveys were complete, determined a weighted 'score' of each service to see where
3122 improvements to their documentation was made.

3123 This paper is structured as thus: Section 7.2 presents related work in the areas
3124 of API usability, intelligent CVSs, and the SUS; Section 7.3 is divided into two
3125 subsections, the first describing how primary sources were selected in a SMS with the
3126 second describing the development of our taxonomy from these sources; Section 7.4
3127 presents the taxonomy; Section 7.5 describes how we developed a survey instrument
3128 of 43 questions to validate the taxonomy against developers, and assess its efficacy
3129 against the three popular CVSs selected to make 12 suggested improvements to
3130 the existing service API documentation; Section 7.6 presents the findings from our
3131 validation analysis and the weightings for the taxonomy; Section 7.7 describes the

3132 threats to validity of this work and Section 7.8 provides concluding remarks and the
3133 future directions of this study. Additional materials are provided in Appendix C.

3134 7.2 Related Work

3135 7.2.1 API Usability and Documentation Knowledge

3136 Use of the SMS approach has explored developer experience and API usability.
3137 A 2018 study reviewed 36 API documentation generation tools and approaches, and
3138 analysed the tools developed and their inputs and documentation outputs [243]. The
3139 findings from this study emphasise that the largest effort in API documentation tool-
3140 ling is to assist developers to generate either example code snippets and/or templates
3141 or natural language descriptions of the API directly from the program’s source code.
3142 These snippets or descriptions can then be placed in the API documentation, thereby
3143 increasing the efficiency at which API documentation can be written. Additionally,
3144 tools from 12 studies target the maintainability of existing APIs of existing APIs,
3145 with tools from 11 studies target the correctness and accuracy of the documentation
3146 by validating that what is written in the documentation is accurate to the technical
3147 structure of the API. From the end-developer’s perspective, some tools (17 studies)
3148 help target improvements to the developer’s understandability and learnability of
3149 new APIs by linking in examples directly with questions such as on Stack Overflow.

3150 However, the results from this study regards the *tooling* used to either assist in
3151 producing, validating or learning from API documentation. While this is a systematic
3152 study with key insights into the types of tooling produced, there is still a gap for a
3153 SMS in what *guidelines* have been produced by the literature in developing natural-
3154 language documentation itself and how well developers *agree* to those guidelines,
3155 which our work has addressed.

3156 Watson [341] performed a heuristic assessment from 35 popular APIs against 11
3157 high-level universal design elements of API documentation. This study highlighted
3158 how many APIs, even popular ones, fail to grasp these basic design elements.
3159 For example, 25% of the documentation sets did not provide any basic overview
3160 documentation to the API. The heuristics used within Watson’s study is based on
3161 only three seminal works and only contains 11 design elements—our study extends
3162 these heuristics and structures them into a consolidated, hierarchical taxonomy which
3163 we then validate against practitioners.

3164 A taxonomy of distinct knowledge patterns within reference documentation
3165 by Maalej and Robillard [208] classified 12 distinct knowledge types. The tax-
3166 onomy was then evaluated against the JDK 6 and .NET 4.0 frameworks, and showed
3167 that the functionality and structure of these APIs are well-communicated, although
3168 core concepts and rationale about the API are quite rarer to see. The authors also
3169 identified low-value ‘non-information’—described as documentation that provides
3170 uninformative boilerplate text with no insight into the API at all—which was sub-
3171 stantially present in the documentation of methods and fields in the two frameworks.
3172 They recommend that developers factor their 12 distinct knowledge types into the
3173 process of code documentation, thereby preventing low-value non-information. The

3174 development of their taxonomy consisted of questions to model knowledge and information,
3175 thereby capturing the reason about disparate information units independent
3176 to context; a key difference to this paper is the systematic taxonomy approach utilised.

3177 7.2.2 Adapting the System Usability Scale

3178 The SUS was first introduced by Brooke as early as 1986 as a “quick and dirty”
3179 survey scale to easily assess the overall usability of a product or service in a timely
3180 manner. Its popularity in the usability community demonstrated the need for a
3181 tool that can collect a quantifiable rating of usability from a participant’s subjective
3182 opinion, and was later published in [55]. Since, its adoption as an industry standard is
3183 widely demonstrated [19, 56] and studies have adopted its ease of use for generalised
3184 purposes.

3185 While translation of the SUS into other languages [43, 213, 290] is generally
3186 the most adapted form of Brooke’s original survey, some studies have proposed
3187 alternative measurement models to the SUS, such as separating the usability and
3188 learnability components of the survey into a two-dimensional structure [43]. Other
3189 adaptations of the SUS include a 2014 study that proposed a usability scale based
3190 on the SUS for Handheld Augmented Reality applications [289] conceptualised
3191 against comprehensibility and manipulability. However, few studies have designed
3192 questionnaires patterned from the SUS in other contexts, and to our knowledge, this
3193 study presents an initial attempt at doing so in the API documentation knowledge
3194 domain.

3195 7.2.3 Computer Vision Services

3196 Recent studies into cloud-based CVSSs have demonstrated that poor reliability and
3197 robustness in computer vision can ‘leak’ into end-applications if such aspects are
3198 not sufficiently appreciated by developers. A study by Hosseini et al. [149] showed
3199 that Google Cloud Vision’s labelling fails when as little as 10% noise is added to the
3200 image. Facial recognition classifiers are easily confused by modifying pixels of a face
3201 and using transfer learning to adapt one person’s face into another [336]. Our own
3202 prior work found that the non-deterministic evolution of these types of services is not
3203 adequately communicated to developers [81], resulting in lost developer productivity
3204 whereby developers ask fundamental questions about the concepts behind these
3205 services, how they work, and where better documentation can be found [84]. This
3206 paper continues this line of research by providing a means for service providers to
3207 better document their services using a taxonomy and suggested improvements.

3208 7.3 Taxonomy Development

3209 We developed our taxonomy under two primary phases. First, we conducted a SMS
3210 identifying API documentation studies, following guidelines by Kitchenham and
3211 Charters [178] and Petersen et al. [260] (Section 7.3.1). A high level overview of
3212 this first phase is given in Figure 7.2. Second, we followed a software engineering
3213 taxonomy development method by Usman et al. [330] (Section 7.3.2) based on the

³²¹⁴ findings of our SMS, which involved an extensive validation involving real-world
³²¹⁵ developers and contextualised with computer vision APIs (Section 7.5).

³²¹⁶ 7.3.1 Systematic Mapping Study

³²¹⁷ 7.3.1.1 Research Questions (RQs)

³²¹⁸ The first step in producing our SMS was to pose two RQs:

- ³²¹⁹ • **RQ1:** What knowledge do API documentation studies contribute?
- ³²²⁰ • **RQ2:** How is API documentation studied?

³²²¹ Our intent behind RQ1 was to collect as many studies provided by literature on how
³²²² API documentation should be written using natural language (i.e., not using assistive
³²²³ tooling). This helped us shape and form the taxonomy provided in Section 7.4.
³²²⁴ Secondly, RQ2's intent was to understand how the studies derive at their conclusions,
³²²⁵ thereby helping us identify gaps in literature where future studies can potentially
³²²⁶ focus.

³²²⁷ 7.3.1.2 Automatic Filtering

³²²⁸ As done in similar software engineering studies [122, 129, 330], we explored au-
³²²⁹ tomatic filtering of online databases. We defined which SWEBOK knowledge
³²³⁰ areas [154] were relevant to devise a search query. Our search query was built using
³²³¹ related knowledge areas, relevant synonyms, and the term 'software engineering'
³²³² (for comprehensiveness) all joined with the OR operator. Due to the lack of a
³²³³ standard definition of an API, we include the terms: 'API' and its expanded term;
³²³⁴ software library, component and framework; and lastly software development kit
³²³⁵ (SDK). These too were joined with the OR operator, appended with an AND. Lastly,
³²³⁶ 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 )
```

³²³⁷ We executed the query on all available metadata (title, abstract and keywords) in
³²³⁸ May 2019 against Web of Science¹ (WoS), Compendex/Inspec² (C/I) and Scopus³.
³²³⁹ We selected three particular primary sources given their relevance in software en-
³²⁴⁰ gineering literature (containing the IEEE, ACM, Springer and Elsevier databases)
³²⁴¹ and their ability to support advanced queries [54, 178]. A total 4,501 results⁴ were
³²⁴² found, with 549 being duplicates. Table 7.1 displays our results in further detail (du-
³²⁴³ plicates not omitted); Figure 7.1 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>.

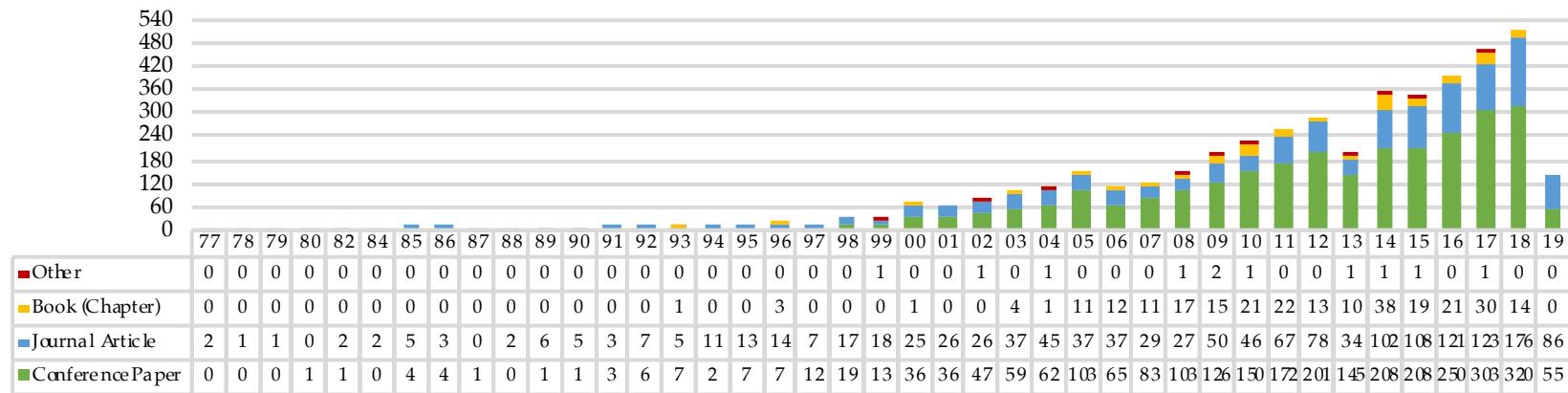


Figure 7.1: Search results by year and venue type.

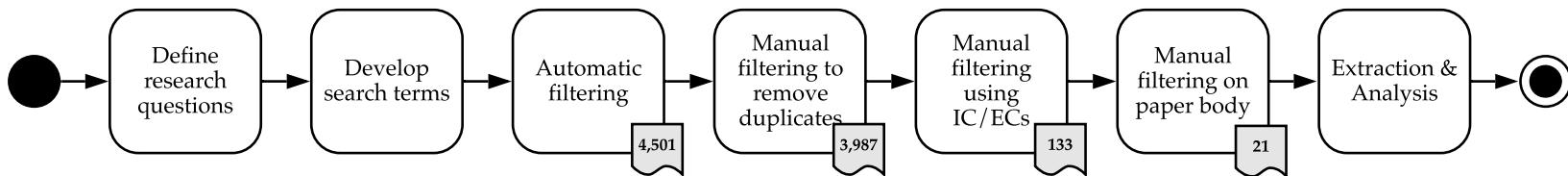


Figure 7.2: A high level overview of the filtering steps from defining and executing our search query to the data extraction of our primary studies. Number of accepted papers resulting from each filtering step is shown.

Table 7.1: 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

3244 publications produced within the last two decades. (As this search was conducted
3245 in May 2019, results taper in 2019.)

3246 7.3.1.3 *Manual Filtering*

3247 A follow-up manual filtering stage followed the 4,501 results obtained by automatic
3248 filtering. As described below, we applied the following inclusion criteria (IC) and
3249 exclusion criteria (EC) to each result:

- 3250 **IC1** Studies must be relevant to API documentation: specifically, we exclude
3251 studies that deal with improving the technical API usability (e.g., improved
3252 usage patterns);
- 3253 **IC2** Studies must propose new knowledge or recommendations to document
3254 APIs;
- 3255 **IC3** Studies must be relevant to software engineering as defined in SWEBOk;
- 3256 **EC1** Studies where full-text is not accessible through standard institutional databases;
- 3257 **EC2** Studies that do not propose or extend how to improve the official, natural
3258 language documentation of an API;
- 3259 **EC3** Studies proposing a third-party tool to enhance existing documentation or
3260 generate new documentation using data mining (i.e., not proposing strategies
3261 to improve official documentation);
- 3262 **EC4** Studies not written in English;
- 3263 **EC5** Studies not peer-reviewed.

3264 Each of these ICs and ECs were applied to every paper after exporting all
3265 metadata of our results to a spreadsheet. The first author then curated the publications
3266 using the following revision process.

3267 Firstly, we read the publication source—to rapidly omit non-software engineer-
3268 ing papers—as well as the author keywords, title, and abstract of all 4,501 studies.
3269 As some studies were duplicated between our three primary sources, we needed to
3270 remove any repetitions. We sorted and reviewed any duplicate DOIs and fuzzy-
3271 matched all very similar titles (i.e., changes due to punctuation between primary
3272 sources), thereby retaining only one copy of the paper from a single database. Sim-
3273ilarly, as there was no limit to our date ranges, some studies were republished in
3274 various venues (i.e., same title but different DOIs). These were also removed using

3275 fuzzy-matching on the title, and the first instance of the paper's publication was
3276 retained. This second phase resulted in 3,987 papers.

3277 Secondly, we applied our inclusion and exclusion criteria to each of the 3,987
3278 papers by reading the abstract. Where there was any doubt in applying the criteria
3279 to the abstract alone, we automatically shortlisted the study. We rejected 427 studies
3280 that were unrelated to software engineering, 3,235 were not directly related to docu-
3281 menting APIs (e.g., to enhance coding techniques that improve the overall developer
3282 usability of the API), 182 proposed new tools to enhance API documentation or
3283 used machine learning to mine developer's discussion of APIs, and 10 were not in
3284 English. This resulted in 133 studies being shortlisted to the final phase.

3285 Thirdly, we re-evaluated each shortlisted paper by re-reading the abstract, the
3286 introduction and conclusion. We removed a further 64 studies that were on API
3287 usability or non API-related documentation (i.e., code commenting). At this stage,
3288 we decided to refine our exclusion criteria to better match the research goals of this
3289 study by including the word 'natural language' documentation in EC2. This removed
3290 studies where the focus was to improve technical documentation of APIs such as
3291 data types and communication schemas. Additionally, we removed 26 studies as
3292 they were related to introducing new tools (EC3), 3 were focused on tools to mine
3293 API documentation, 7 studies where no recommendations were provided, 2 further
3294 duplicate studies, and a further 10 studies where the full text was not available,
3295 not peer reviewed or in English. Books are commonly not peer-reviewed (EC5),
3296 however no books were shortlisted within these results. This final stage resulted in
3297 21 primary studies for further analysis, and the mapping of primary study identifiers
3298 to references S1–21 can be found in Appendix C.3.

3299 As a final phase, we conducted reliability analysis of our shortlisting method.
3300 We conducted intra-rater reliability of our 133 shortlisted papers using the test-
3301 retest approach suggested by Kitchenham and Charters [178]. We re-evaluated a
3302 random sample of 10% of the 133 shortlisted papers a week after initial studies were
3303 shortlisted. This resulted in *substantial agreement* [192], measured using Cohen's
3304 kappa ($\kappa = 0.7547$).

3305 7.3.1.4 Data Extraction & Systematic Mapping

3306 Of the 21 primary studies, we conducted abstract key-wording adhering to Petersen
3307 et al.'s guidelines [260] to develop a classification scheme. An initial set of keywords
3308 were applied for each paper in terms of their methodologies and research approaches
3309 (RQ2), based on an existing classification schema used in the requirements engineer-
3310 ing field by Wieringa et al. [347]. These are: *evaluation papers*, which evaluates
3311 existing techniques in-practice; *validation papers*, which investigates proposed tech-
3312 niques not yet implemented in-practice; *experience papers*, which do investigate or
3313 evaluate either proposed or existing techniques, but presents insightful experiences
3314 of authors that warrant communication to other practitioners; and *philosophical pa-
pers*, which presents new conceptual frameworks that describes a language by which
3316 we can describes our observations of existing or new techniques, thereby implying
3317 a new viewpoint for understanding phenomena.

Table 7.2: 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

3318 After all primary studies had been assigned keywords, we noticed that all papers
 3319 used field study techniques, and thus we consolidated these keywords using Singer
 3320 et al.'s framework of software engineering field study techniques [305]. Singer et al.
 3321 captures both study techniques *and* methods to collect data within the one framework,
 3322 namely: *direct techniques*, including brainstorming and focus groups, interviews and
 3323 questionnaires, conceptual modelling, work diaries, think-aloud sessions, shadowing
 3324 and observation, participant observation; *indirect techniques*, including instrumenting
 3325 systems, fly-on-the-wall; and *independent techniques*, including analysis of work
 3326 databases, tool use logs, documentation analysis, and static and dynamic analysis.

3327 Table 7.2 describes our data extraction form, which was used to collect relevant
 3328 data from each paper. Figure 7.3 presents our systematic mapping, where each study
 3329 is mapped to one (or more, if applicable) of methodologies plotted against Wieringa
 3330 et al.'s research approaches. We find that a majority of these studies survey develop-
 3331 ers using direct techniques (i.e., interviews and questionnaires) and some performing
 3332 structured documentation analysis. Few studies report recent experiences, with the
 3333 majority of API documentation knowledge being evaluation research, and some val-
 3334 idation studies. There are few experience papers describing anecdotal evidence of
 3335 API documentation knowledge, and almost no philosophical papers that describe new
 3336 conceptual ways at approaching API documentation as a large majority of existing
 3337 work either evaluates existing (in-practice) strategies or validates the effectiveness
 3338 of new strategies.

3339 **7.3.2 Development of the Taxonomy**

3340 A majority of taxonomies produced in software engineering studies are often made
 3341 extemporaneously [330]. For this reason, we decided to proceed with a systematic
 3342 approach to develop our taxonomy using the guidelines provided by Usman et al.
 3343 [330], which are extended from lessons learned in more mature domains. In this
 3344 subsection, we outline the 4 phases and 13 steps taken to develop our taxonomy

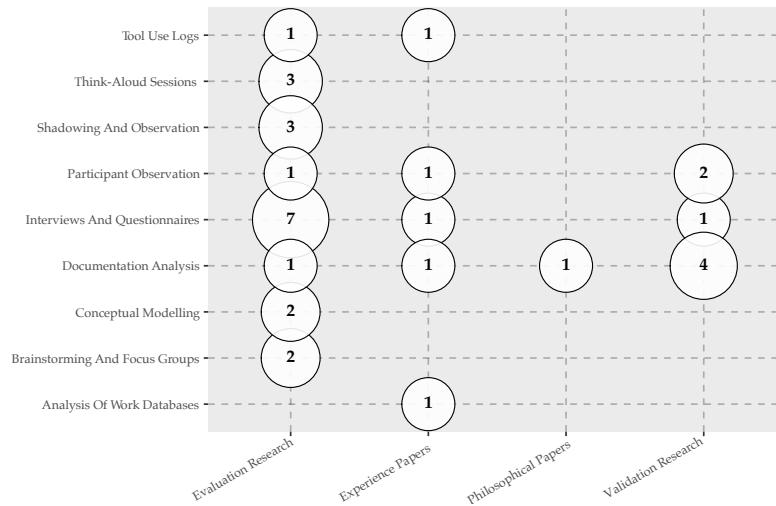


Figure 7.3: Systematic map: field study technique vs research type

³³⁴⁵ based on Usman et al.'s technique. Usman et al.'s final *validation* phase is largely
³³⁴⁶ detailed within Section 7.5 after we present our taxonomy in Section 7.4.

³³⁴⁷ 7.3.2.1 Planning phase

³³⁴⁸ The preliminary phase involves answering the following:

- ³³⁴⁹ **(1) define the software engineering knowledge area:** The software engineering
³³⁵⁰ knowledge area, as defined by the SWEBOK, is software construction;
- ³³⁵¹ **(2) define the objective:** The main objective of the proposed taxonomy is to define
³³⁵² a set of categories that enables to classify different facets of natural-language
³³⁵³ API documentation knowledge (not API *usability* knowledge) as reported in
³³⁵⁴ existing literature;
- ³³⁵⁵ **(3) define the subject matter:** The subject matter of our proposed taxonomy is
³³⁵⁶ documentation artefacts of APIs;
- ³³⁵⁷ **(4) define the classification structure:** The classification structure of our proposed
³³⁵⁸ taxonomy is *hierarchical*;
- ³³⁵⁹ **(5) define the classification procedure:** The procedure used to classify the docu-
³³⁶⁰ mentation artefacts is qualitative;
- ³³⁶¹ **(6) define the data sources:** The basis of the taxonomy is derived from field study
³³⁶² techniques (see Section 7.3.1.4).

³³⁶³ 7.3.2.2 Identification and extraction phase

³³⁶⁴ The second phase of the taxonomy development involves **(7) extracting all terms**
³³⁶⁵ and **concepts** from relevant literature, which we have achieved from our SMS. These
³³⁶⁶ terms are then consolidated by **(8) performing terminology control**, as some terms
³³⁶⁷ may refer to different concepts and vice-versa.

3368 7.3.2.3 *Design phase*

3369 The design phase identified the core dimensions and categories within the extracted
3370 data items. The first step is to (9) *identify and define taxonomy dimensions*; for this
3371 study we utilised a bottom-up approach to identify each dimension, i.e., extracting the
3372 categories first and then nominating which dimensions these categories fit into using
3373 an iterative approach. As we used a bottom-up approach, step (9) also encompassed
3374 the second stage of the design phase, which is to (10) *identify and describe the*
3375 *categories of each dimension*. Thirdly, we (11) *identify and describe relationships*
3376 between dimensions and categories, which can be skipped if the relationships are
3377 too close together, as is the case of our grouping technique which allows for new
3378 dimensions and categories to be added. The last step in this phase is to (12) *define*
3379 *guidelines for using and updating the taxonomy*. The taxonomy is as simple as a
3380 checklist that can be heuristically applied to an API document, and each dimension
3381 is malleable and covers a broad spectrum of artefacts; while we do not anticipate
3382 any further dimensions to be added, new categories can easily be fitted into one of
3383 the dimensions (see Section 7.8). We provide guidelines for use in our application
3384 of the taxonomy against CVSSs within Sections 7.4 and 7.6.

3385 7.3.2.4 *Validation phase*

3386 In the final phase of taxonomy development, taxonomy designers must (13) *validate*
3387 *the taxonomy* to assess its usefulness. Usman et al. [330] describe three approaches to
3388 validate taxonomies: (i) orthogonal demonstration, in which the taxonomy's orthog-
3389 onality is demonstrated against the dimensions and categories, (ii) benchmarking
3390 the taxonomy against similar classification schemes, or (iii) utility demonstration by
3391 applying the taxonomy heuristically against subject-matter examples. In our study,
3392 we adopt utility demonstration by use of a survey and heuristic application of the
3393 taxonomy against real-world case-studies (i.e., within the domain of CVSSs). This is
3394 is discussed in greater detail within Section 7.5.

3395 7.4 API Documentation Knowledge Taxonomy

3396 Our taxonomy consists of five dimensions (labelled A–E). We expand these five di-
3397 mensions into 34 categories (sub-dimensions). Each dimension respectively covers:

- 3398 • [A] **Usage Description** on *how* to use the API for the developer's intended
3399 use case;
- 3400 • [B] **Design Rationale** on *when* the developer should choose this API for a
3401 particular use case;
- 3402 • [C] **Domain Concepts** of the domain behind the API to understand *why* this
3403 API should be chosen for this domain;
- 3404 • [D] **Support Artefacts** that describe *what* additional documentation the API
3405 provides; and
- 3406 • [E] **Documentation Presentation** to help organise the *visualisation* of the
3407 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 7.4: Our proposed taxonomy on what artefacts should be documented in a complete API document.

³⁴⁰⁸ Further descriptions of the categories encompassing each dimension are given within
³⁴⁰⁹ Figure 7.4 and Appendix C.1, coded as $[Xi]$, where i is the category identifier within
³⁴¹⁰ a dimension, $X \in \{A, B, C, D, E\}$.

³⁴¹¹ Appendix C.1 shows which of the primary sources (S1–21) provide the rec-
³⁴¹² ommendation described as well as an ‘in-literature score’ (ILS). This score is a
³⁴¹³ weighting calculated as a percentage of the number of primary studies that make the
³⁴¹⁴ recommendation divided by the total of primary studies, and indicates the overall
³⁴¹⁵ level of agreement that academic sources suggest these documentation artefacts.
³⁴¹⁶ This score is contrasted to the ‘in-practice score’ (IPS) which indicates the over-
³⁴¹⁷ all level of agreement that *practitioners* think such documentation artefacts are
³⁴¹⁸ needed. Further details about the ILS and IPS values, how they were calculated and
³⁴¹⁹ analysed for each category, and a rigorous contrast between the two are provided
³⁴²⁰ Sections 7.5.1.2 and 7.6.1 to 7.6.3. For comparative purposes, we illustrate a colour
³⁴²¹ scale (from red to green) to indicate the relevancy weight between ILS and IPS
³⁴²² values in Appendix C.1: for example, while quick-start guides [A1] are few refer-
³⁴²³ enced in academic sources at 14%, they are generally well-desired by practitioners
³⁴²⁴ 88% agreement. We then provide three columns that assesses the presence of these
³⁴²⁵ documentation artefacts against three popular CVSSs: Google Cloud Vision, AWS’s
³⁴²⁶ Rekognition, and Azure Cloud Vision (abbreviated to GCV, AWS and ACV). A
³⁴²⁷ fully shaded circle (●) indicates that the documentation artefact was clearly found
³⁴²⁸ in the service, while a half-shaded circle (◐) indicates that the artefact was only
³⁴²⁹ partially present. An outlined circle (○) indicates that the service lacks the indicated
³⁴³⁰ documentation artefact within our taxonomy. This empirical assessment is further
³⁴³¹ detailed in Section 7.6.5, which outlines concrete areas in the respective services’
³⁴³² documentation where improvements could be made, as well as hyperlinks to the
³⁴³³ documentation where relevant.

³⁴³⁴ Figure 7.4 illustrates these findings, with underlines indicating key artefacts and
³⁴³⁵ various iconography to indicate specific results. The computer icon (💻) includes a
³⁴³⁶ ranking from 1–5 of the top five most recommended artefacts according to devel-
³⁴³⁷ opers, as calculated from their relevant IPS scores. Conversely, the book icon (📖)
³⁴³⁸ indicates the rankings of the top five most recommended artefacts according to liter-
³⁴³⁹ ature. For example, while literature suggests the most useful documentation artefact
³⁴⁴⁰ are API usage description code snippets [A5], in-practice, we find that developers
³⁴⁴¹ prefer design rationale on what the limitations of API are [B7] with code snippets
³⁴⁴² coming in second place. Where there is strong agreement between developers and
³⁴⁴³ literature (within a standard deviation of 0.15) we use the handshake icon (🤝) and
³⁴⁴⁴ list whether both agree if the category is Very Helpful (VH), Slightly Helpful (SH) or
³⁴⁴⁵ Not Helpful (NH). Further details on this explanation are provided in Section 7.6.3.
³⁴⁴⁶ Lastly, we provide iconography for the presence (✓) or non-presence (✗) of these
³⁴⁴⁷ artefacts in *all three* CVSSs assessed, per Section 7.6.2.

3448 7.5 Validating our Taxonomy

3449 7.5.1 Survey Study

3450 7.5.1.1 Designing the Survey

3451 We followed the guidelines by Kitchenham and Pfleeger [179] on conducting personal opinion surveys in software engineering to validate our survey. In developing
3452 our survey instrument, we shaped questions around each of our 5 dimensions and
3453 34 categories. To achieve this, we used Brooke's SUS [55] as inspiration and re-
3454 shaped the 34 categories around a question. Each dimension was marked a numeric
3455 question (3–7), and alphabetic sub-questions were marked for each sub-dimension
3456 or category.

3457 We used closed questioning where respondents could choose an answer on a
3458 5-point Likert-scale (1=*strongly disagree*, 2=*somewhat disagree*, 3=*neither agree*
3459 nor *disagree*, 4=*slightly agree* and 5=*strongly agree*). Like Brooke's study, each
3460 question alternated in positive and negative sentiment. Half of our questions were
3461 written where a likely common response would be in strong agreement and vice-
3462 versa for the other half, such that participants would have to “read each statement
3463 and make an effort to think whether they would agree or disagree with it” [55]. For
3464 example, the question regarding [B7] on API limitations was framed as: “*I believe it*
3465 *is important to know about what the limitations are on what the API can and cannot*
3466 *provide*” (Q4g), whereas the question regarding [C1] on domain concepts of the API
3467 was framed as: “*I wouldn't read through theory about the API's domain that relates*
3468 *theoretical concepts to API components and how both work together*” (Q5a).

3469 In addition, the remaining eight questions asked demographical information.
3470 An extra open question asked for further comments. The full survey is provided in
3471 Appendix C.4.

3473 7.5.1.2 Evaluating the Survey

3474 After the first pass at designing questions was completed, we evaluated our survey
3475 on three researchers within our research group for general feedback. This resulted
3476 in minor changes, such as slight re-wording of questions, clarifying the difference
3477 between web services and web APIs, and providing specific questions with examples
3478 (some with images). For example, the question regarding [A9] on an exhaustive list
3479 of all major components in the API was framed as “*I believe an exhaustive list*
3480 *of all major components in the API without excessive detail would be useful when*
3481 *learning an API*” (Q3i) with the example “e.g., a computer vision web API might
3482 *list object detection, object localisation, facial recognition, and facial comparison*
3483 *as its 4 components*”.

3484 After this, we conducted reliability analysis using a test-retest approach on three
3485 developers within our group seven weeks apart. This was calculated using the `irr`
3486 computational R package [119] (as suggested in [137]) and resulted in an average
3487 intra-class correlation of 0.63 which indicates a good overall index of agreement [72].

3488 *7.5.1.3 Recruiting Participants*

3489 Our target population for the study was application software developers with varying
3490 degrees of experience (including those who and who have not used CVSs or related
3491 tools before) and varying understanding of fundamental machine learning concepts.
3492 We began by recruiting software developers within our research group using a
3493 group-wide message sent on our internal messaging system. Of the 44 developers in
3494 our group's engineering cohort, 22 responses were returned, indicating an internal
3495 response rate of 50%.

3496 For external participant recruiting, we shared the survey on social media plat-
3497 forms and online-discussion forums relevant to software development. We adopted
3498 a non-probabilistic snowballing sampling where the participants, at the end of the
3499 survey, were encouraged to share the survey link to others using *AddThis*⁵. This
3500 resulted in 43 additional visits to the survey. Additionally, snowballing sampling was
3501 encouraged within members of our research group who shared the survey with an
3502 additional 21 participants. However, while there were a total of 86 respondents, only
3503 51 finished the survey, leaving 35 participants with partially completed responses.
3504 Our final response rate was therefore 59%, which is very close to median response
3505 rates of 60% [24] in information systems and 5% in software engineering [305].

3506 *7.5.1.4 Analysing Response Data*

3507 To analyse our response data, we used an adapted version of the SUS method to
3508 produce a score for each question's 5-point response. As per Brooke's methodology,
3509 we mapped the responses from their ordinal scale of 1–5 to 0–4, and subtracted that
3510 value by 1 for positive questions and subtracted the value from 5 for the negative
3511 questions [55]. Unlike Brooke's method, we averaged each response for every
3512 question and divided by four (i.e., now a 4-point scale) to obtain scores for each
3513 category. This is presented in Appendix C.1 under the 'in-practice score' (IPS) for
3514 each category.

3515 Demographics for our survey were consistent in terms of the experience levels of
3516 developers who responded. Most were professional programmers with 75% report-
3517 ing between 1–10 years of work experience. A majority of our respondents (33%)
3518 reported to be in mid-tier roles. Most worked in either consulting or information
3519 technology services, reported at 17% for both.

3520 **7.5.2 Empirical application of the taxonomy against Computer Vision
3521 Services**

3522 Once our taxonomy had been developed, we performed an empirical application
3523 against three CVSs: Google Cloud Vision [388], Amazon Rekognition [363] and
3524 Azure Computer Vision [402]. Our selection criteria in choosing these particular
3525 services to analyse is based on the prominence of the service providers in industry
3526 and the ubiquity of their cloud platforms (Google Cloud, Amazon Web Services,
3527 and Microsoft Azure) in addition to being the top three adopted vendors used for

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

3528 cloud-based enterprise applications [277]. In addition, we had conducted extensive
3529 investigation into the services' non-deterministic runtime behaviour and evolution
3530 profile in prior work [81] and have also identified developers' complaints about their
3531 incomplete documentation in a prior mining study on Stack Overflow [84].

3532 We began with an exploratory analysis of the presence of each dimension and
3533 its categories. Appendix C.2 displays all sources of documentation used; although
3534 we initially started on the respective services homepages [363, 388, 402], this search
3535 was expanded to other webpages hyperlinked. For each category, we listed the
3536 documentation's presence as either fully present, partially present or not present
3537 at all. This is shown in Appendix C.1 with the indication of (half-)filled circles or
3538 circle outlines for Google Cloud Vision (abbreviated to GCV), Amazon Rekognition
3539 (abbreviated to AWS), and Azure Computer Vision (abbreviated to ACV). Notes were
3540 taken for each webpage justifying the presence, and exact sources of documentation
3541 were listed when (partially) present. PDFs of each webpage were downloaded
3542 between 14–18 March 2019 for analysis.

3543 Once our analysis was completed and results from the survey finalised, we then
3544 calculated *weighted* ILS and IPS values for each dimension's category. This was done
3545 by multiplying the ILS and IPS values for each category (listed in Appendix C.1) by
3546 either 0, 0.5 or 1 for categories not present, partially present, or present (respectively)
3547 in each service. The 'maximum' ILS and IPS values indicate the highest possible
3548 score a service can be ranked as though *all* categories are present. Tables 7.3 and 7.4
3549 show the sum of weights for each category in its respective dimension, in addition to
3550 the maximum possible score. Again, we use the same abbreviations for each service
3551 as per Appendix C.1. The scores are normalised into percentages for comparative
3552 purposes as a ratio of the score over all dimensions for a particular service to
3553 the maximum possible score. For comparative purposes, these are illustrated in
3554 Figure 7.6.

3555 7.6 Taxonomy Analysis

3556 In this section, we analyse investigating the taxonomy from two perspectives. Firstly,
3557 we describe the ILS values, being an interpretation of the number of papers that
3558 conclude the recommendations in each category and dimension, and the weighted ILS
3559 scores, being an application of the taxonomy specifically to CVSs. Secondly, we look
3560 at the results from our survey and their respective IPS values, being an interpretation
3561 of how well developers agree with these recommendations, and the weighted IPS
3562 scores, being the application of how application developers would agree with the
3563 documentation of the CVSs. We then contrast the difference between what literature
3564 recommends and how well developers agree with these recommendations.

3565 7.6.1 In-Literature Scores for Taxonomy Categories

3566 ILS values indicate the proportion of papers that recommend categories within our
3567 taxonomy of all 21 studies. The most highly recommended categories from our
3568 SMS fall under the Usage Description dimension. The majority (0.71) of studies

Table 7.3: 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 7.4: 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

³⁵⁶⁹ advocate for code snippets as a necessary piece in the API documentation puzzle ³⁵⁷⁰ [A5]. While code snippets generally only reflect small portions of API functionality ³⁵⁷¹ (limited to 15–30 LoC), this is complimented by step-by-step tutorials (0.57) that tie ³⁵⁷² in multiple (disparate) components of API functionality, generally with some form ³⁵⁷³ of screenshots, demonstrating the development of a non-trivial application using the ³⁵⁷⁴ API step-by-step [A6]. The third highest category scored was also under the Usage ³⁵⁷⁵ Description dimension, being low-level reference documentation at 0.52 [A2]. These ³⁵⁷⁶ three categories were the only categories to be scored as majority categories (i.e., ³⁵⁷⁷ their scores were above 0.50). The fourth and fifth highest scores are an entry- ³⁵⁷⁸ level purpose/overview of the API (0.48) that gives a brief motivation as to why a ³⁵⁷⁹ developer should choose a particular API over another [B1] and consistency in the ³⁵⁸⁰ look and feel of the documentation throughout all of the API’s official documentation ³⁵⁸¹ (0.43) [E6].

³⁵⁸² 7.6.2 In-Practice Scores for Taxonomy Categories

³⁵⁸³ IPS values indicate the extent to which developers ‘agree’ with the statements made ³⁵⁸⁴ in our survey, as calculated using the SUS technique [55]. These values are generally ³⁵⁸⁵ greater than the ILS values, since they are ranked by all survey participants and are not ³⁵⁸⁶ a ratio of the 21 primary studies. Unlike ILS scores, 28 categories scored above 0.50. ³⁵⁸⁷ The highest dimension corroborates that of the ILS scores; within the top five ranked ³⁵⁸⁸ ILS scores, Usage Description categories feature four times. However, developers ³⁵⁸⁹ generally find limitations on what the APIs can and cannot provide the most useful, ³⁵⁹⁰ at 0.94, which falls under the Design Rationale dimension [B7]. Following this, the

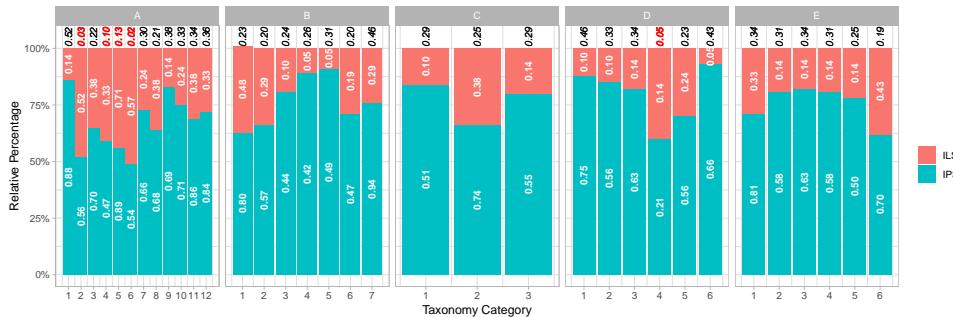


Figure 7.5: Comparison of ILS and IPS values for each category (grouped by dimensions) presented as a relative percentage.

Table 7.5: 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

3591 code-snippets [A5] is highly ranked (as per the ILS values) with developers agreeing
 3592 that code-snippets should be included in most API documentation. Quick-start
 3593 guides [A1] are the next most-useful category that developers advocate for, reported
 3594 at 0.88. Following this, the instructions on how to install the API or begin using the
 3595 API, its release cycle, and frequently it is updated [A11] is also important, ranking
 3596 fourth at 0.86. Lastly, error definitions describing how developers can address
 3597 problems [A12] were scored at 0.84.

3598 7.6.3 Contrasting In-Literature to In-Practice Scores

3599 Figure 7.5 highlights the relative percentage of each ILS and IPS value for all
 3600 subcategories, thereby indicating the relative agreement between the two. In this
 3601 graph, an ILS and/or IPS core approaching a relative percentage of 50% indicates
 3602 equal agreement whereby both developer's and literary references share a similar
 3603 distribution of recommendation agreement. Italicised labels above each column
 3604 indicates the standard deviation between the ILS and IPS values, where red labels
 3605 indicated a standard deviation less than 0.15 (i.e., developers and literature agree to
 3606 the values to a similar extent).

3607 Where the standard deviation between ILS and IPS values is less than 0.015
 3608 (as indicated by red labels above each column in Figure 7.5), then there is strong
 3609 alignment between both scores. However, of all 34 categories, only five cases of this
 3610 occur. Developers agree to the academic works that make the recommendations *to*
 3611 *the same relative proportion* as per the labels assigned in Table 7.5:

- 3612 • Having email addresses or phone numbers listed within an API is generally

3613 not helpful at all [D4],

- 3614 • Introspecting the source code comments of an API is only somewhat helpful [A4],
- 3615 • Low-level reference documentation with all objects and methods (etc.) documented is slightly helpful [A2],
- 3616 • Following step-by-step tutorials are also slightly helpful [A6],
- 3617 • Code snippets are the most helpful [A5].

3620 The remaining categories in the dimension do not share strong association between both developer opinions and the number of papers producing recommendations. Due to the disparity between these ILS and IPS values, we do not report on 3623 their utility.

3624 7.6.4 Triangulating ILS and IPS with Computer Vision

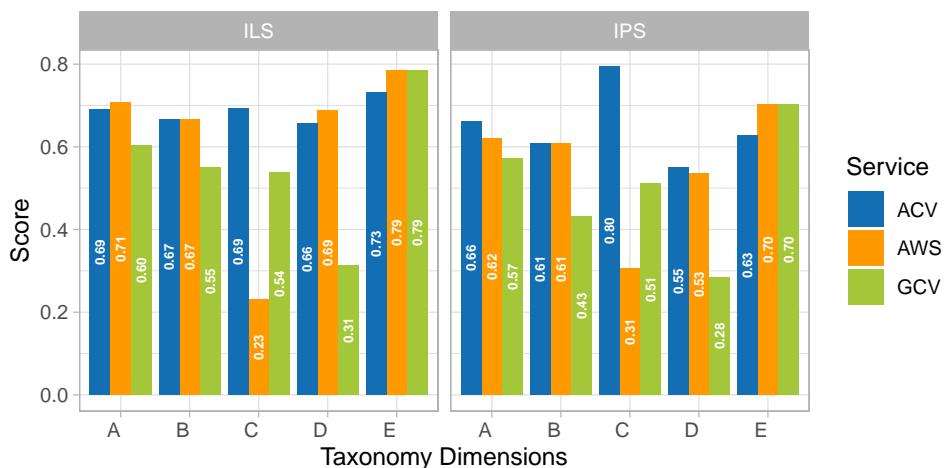


Figure 7.6: Comparison of the weighted ILS and IPS values for the three CVSs assessed.

3625 When applied in the context of CVSs, we see that Azure Computer Vision (ACV) and Amazon Rekognition (AWS) are better documented than Google Cloud 3626 Vision (GCV), particularly in Design Rationale and Usage Description. Figure 7.6 highlights that Azure Computer Vision is especially well documented in Domain 3627 Concepts when measured using the weighted ILS and has the highest score of all services and dimensions. It is evident that Google Cloud Vision needs improved 3628 Design Rationale documentation and further Support Artefacts would be helpful. 3629 Generally speaking, Google Cloud Vision is less ‘complete’ than other services, 3630 except in Domain Concepts documentation and its Documentation Presentation.

3631 In the context of CVSs, IPS values share a similar distribution to ILS values. Notably, in-practice, it seems developers prefer the documentation of Amazon 3632 Rekognition compared to the the in-literature weighted scoring of Azure Computer 3633 Vision (Figure 7.6). Except in the case of documenting Domain Concepts, Amazon 3634

3638 Rekognition scores slightly higher than Azure Computer Vision except for the De-
3639 sign Rationale documentation where it is equal. Similar to the ILS scoring, Google
3640 Computer Vision has low compliance to the recommendations proposed, except in
3641 its Documentation Presentation.

3642 7.6.5 Areas of Improvement for CVS Documentation

3643 Triangulating the taxonomy developed from literary sources, the developer survey
3644 on this taxonomy to understand its efficacy in-practice, and applying the taxonomy to
3645 the CVS domain, we are able to assess the key areas of improvement in this domain.

3646 For this assessment, we select the ILS or IPS values for categories that are
3647 considered either somewhat or very helpful (i.e., a score greater than 0.50). We then
3648 match these against categories that are found to be partially or not present within
3649 each service. In total, we found 12 categories where improvements can be made
3650 across all dimensions except Documentation Presentation, detailed below .

3651 7.6.5.1 Issues regarding Usage Description

3652 **Quick-start guides [A1]:** Quick-start guides should provide a short tutorial that
3653 allows programmers to pick up the basics of an API in a programming language of
3654 their choice. For the services assessed, each offer various client SDKs (e.g., as Java
3655 or Python client libraries). Google Cloud Vision and Azure Computer Vision offer
3656 quick-start guides [391, 409] in which sets of articles target various SDKs or are
3657 client-agnostic with code snippets that can be changed to the client language/SDK
3658 of the developer's choice. Amazon Rekognition offers exercises in setting up the
3659 AWS SDK and using the command-line interface to interact with image analysis
3660 components [369], however this is client-agnostic nor does it provide details in how
3661 to get started with using the client SDKs.

3662 **💡 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'.

3663 **Step-by-step tutorials [A6]:** Google Cloud Vision offers tutorials limited to one
3664 component. These do not sufficiently demonstrate how to combine *multiple components* of the API together and how developers should integrate it with a differ-
3665 ent platform, which a good step-by-step tutorial should detail. The official AWS
3666 Machine Learning blog [366] provides extensive tutorials (in some cases, with a
3667 suggested tutorial completion time of over an hour) that integrate multiple Amazon
3668 Rekognition components with other AWS components. Microsoft provide tutori-
3669 als [407, 412, 413] integrating multiple components within their service to mobile
3670 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.

3671 **Downloadable production-ready applications [A7]:** Microsoft provide a down-
3672 loadable application [411] that explores many components of the Azure Computer
3673 Vision API. The application is thoroughly documented with and also provides guid-
3674 ance on how to structure the architecture design of the program. While Rekognition
3675 and Google Cloud Vision also provide downloadable source code, they are largely
3676 under-documented, do not combine multiple components of the API together, and
3677 only use god-classes to handle all requests to the API [370, 393].

 **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.

3678 **Understanding best-practices [A8]:** Google Cloud provides best-practices for its
3679 platform in both general and enterprise contexts [385, 394], but there is little advice
3680 provided to guide developers on how best to use Google Cloud Vision. Microsoft
3681 provides guidance on improving results of custom vision classifiers [408], but no
3682 further details on non-custom vision classifiers are found. We found the most detailed
3683 best-practices to be provided by Amazon Rekognition [368], which outlines more
3684 detailed strategies such as reducing data transfer by storing and referencing images
3685 on S3 Buckets or the attributes images should have in various scenarios (e.g., the
3686 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.

3687 **Exhaustive lists of all major API components [A9]:** Amazon provides a two-fold
3688 feature list that describes both the key features of Rekognition at a high-level [367]
3689 as well as a detailed, technical breakdown of each API operation provided within the
3690 service [365]. Microsoft also provide a list of high-level features that Azure Com-
3691 puter Vision can analyse [414] which provides hyperlinks to detailed descriptions of
3692 each feature. Google's Cloud Vision API provides a partial breakdown of the types
3693 of services provided, however this list is not fully complete, nor are there hyperlinks
3694 to more detailed descriptions of each of the features [395].

☞ **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.

3695 **Minimum system requirements and dependencies [A10]:** Although there is no
3696 dedicated webpage for this on any of the services investigated, there are listed
3697 dependencies for the client libraries in Google's and Azure's quick-start guides [391,
3698 405]. These may be embedded within the quick-start guide as developers are likely
3699 to encounter dependency issues when they first start using the API. We found it a
3700 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.

3701 **Installation and release cycle notes [A11]:** It is imperative that developers know
3702 what has changed between releases and how frequently the releases are exported.
3703 We found release notes for Amazon Computer Vision, although they are only major
3704 releases and have not been updated since 2017 [364] which does not account for
3705 evolution in the service's responses [81]. Google's and Microsoft's release notes are
3706 generally more frequently updated, therefore developers can get a sense of its release
3707 frequency [392, 410]. However, there are evolution issues that are not addressed.
3708 Installation instructions are detailed within Rekognition's developer guide, outlining
3709 how to sign up for an account, and install the AWS command-line interface [372].

☞ **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.

3710 7.6.5.2 Issues regarding Design Rationale

3711 **Limitations of the API [B7]:** The most detailed limitations documented were
3712 found on Rekognition's dedicated limitations page [371] that outlines functional
3713 limitations such as the maximum number of faces or words that can be detected
3714 in an image, the size requirements of images, and file type information. For the
3715 other services, functional limitations are generally found within each endpoint's API
3716 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.

³⁷¹⁷ 7.6.5.3 *Issues regarding Domain Concepts*

³⁷¹⁸ **Conceptual understanding of the API [C1]:** Azure Computer Vision provides
³⁷¹⁹ ‘concept’ pages describing the high-level concepts behind computer vision and where
³⁷²⁰ these functions are implemented within the APIs (e.g., [406]). We were unable to
³⁷²¹ 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.

³⁷²² **Definitions of domain-specific terminology [C2]:** Terminologies relevant to ma-
³⁷²³ chine learning concepts powering these CVSs are well detailed within Google’s
³⁷²⁴ machine learning glossary [389], however few examples matching computer vision
³⁷²⁵ are immediately relevant. While this page is linked from the original Google Cloud
³⁷²⁶ Vision documentation, it may be too technical for application developers to grasp. A
³⁷²⁷ slightly better example of this is [414], where developers can understand computer
³⁷²⁸ 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.

³⁷²⁹ 7.6.5.4 *Issues regarding Support Artefacts*

³⁷³⁰ **Troubleshooting suggestions [D2]:** The only troubleshooting tips found in our
³⁷³¹ analysis were in Rekognition’s video service [373]. Further detailed instances of
³⁷³² these troubleshooting tips could be expanded to non-video issues. For instance,
³⁷³³ if developers upload ‘noisy’ images, how can they inform the system of a specific
³⁷³⁴ ontology to use or to focus on parts of the foreground or background of the image?
³⁷³⁵ These are suggestions which we have proposed in prior work [81] that do not seem
³⁷³⁶ to be documented.

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

³⁷³⁷ **Diagrammatic overview of the API [D3]:** None of the CVSs provide any overview
³⁷³⁸ of the API in terms of the features and processing steps on how they should be used.
³⁷³⁹ For instance, pre-processing and post-processing of input and response data should
³⁷⁴⁰ be considered and an understanding of how this fits into the ‘flow’ of an application
³⁷⁴¹ 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.

3742 7.7 Threats to Validity

3743 7.7.1 Internal Validity

3744 Threats to *internal validity* represent internal factors of our study which affect
3745 concluded results. Kitchenham and Charters' guidelines on producing systematic
3746 reviews [178] suggest that researchers conducting reviews should discuss the review
3747 protocol, inclusion decisions, data extraction with a third party. Within this study,
3748 we discussed our protocols with other researchers within our research group and
3749 utilised test-retest reliability. Further assessments into reliability would involve an
3750 assessment of the review and extraction processes, which can be investigated using
3751 inter-rater reliability measures. Guidelines suggested by Garousi and Felderer [121]
3752 describe methods for independent analysis and conflict resolution could help resolve
3753 this.

3754 As stated in Section 7.3.2, we utilised a systematic software engineering tax-
3755 onomy development method by Usman et al. [330]. Two additional taxonomy
3756 validation approaches proposed by Usman et al. were not considered in our work:
3757 benchmarking and orthogonality demonstration. To our knowledge, there are no
3758 other studies that classify existing API knowledge studies into a structured taxon-
3759 omy, and therefore we are unable to benchmark our taxonomy against others. We
3760 would encourage the research community to conduct a replication of our work and
3761 investigate whether our taxonomy classification approaches are replicable to ensure
3762 that categories are reliable and the dimensions fit the objectives of the taxonomy.
3763 Moreover, we did not investigate orthogonality demonstration as our primary goals
3764 for this work were to investigate the efficacy of the taxonomy by practitioners and
3765 in-practice, with reference to our wider research area of intelligent CVSs. Therefore,
3766 we solely adopted the utility demonstration approach in two detailed experiments
3767 (Sections 7.5 and 7.6) to analyse the efficacy of our taxonomy and identify potential
3768 improvements for these services' API documentation.

3769 7.7.2 External Validity

3770 Threats to *external validity* concern the generalisation of our observations. Our
3771 systematic mapping study has used a broad range of sources however not all papers
3772 contributing to API documentation may have been found or captured within the
3773 taxonomy. While we attempted to include as many papers as we could find in our
3774 study, some papers may have been filtered out due to our exclusion criteria. For
3775 example, there are studies we found that were excluded as they were not written in
3776 English, and these excluding factors may alter our conclusions, introducing conflict-

3777 ing recommendations. However, given the consistency of these trends within the
3778 studies that were sourced, we consider this a low likelihood.

3779 Documentation of web APIs are non-static, and may evolve using contributions
3780 from both official sources and the developer community (e.g., via GitHub). We
3781 downloaded the three service’s API documentation in March of 2019—it is highly
3782 likely that new documentation may have been added since or modified since publi-
3783 cation. A recommendation to mitigate this would be to re-evaluate this study once
3784 intelligent CVSs have matured and become even more mainstream in developer
3785 communities.

3786 We also adopt research conducted in the field of questionnaire design, such as
3787 ensuring all scales are worded with labels [188] and have used a summatting rating
3788 scale [310] to address a specific topic of interest if people are to make mistakes in
3789 their response or answer in different ways at different times. This approach was
3790 also extended using the SUS methodology, in which positive and negative items
3791 were used—as multiple studies have shown [56, 290], this approach helps reduce
3792 poor-quality responses by minimising extreme responses and acquiescence biases.

3793 7.7.3 Construct Validity

3794 Threats to *construct validity* relates to the degree by which the data extrapolated
3795 in this study sufficiently measures its intended goals. Automatic searching was
3796 conducted in the SMS by choice of three popular databases (see Section 7.3.1).
3797 As a consequence of selecting multiple databases, duplicates were returned. This
3798 was mitigated by manually curating out all duplicate results from the set of studies
3799 returned. Additionally, we acknowledge that the lack manual searching of papers
3800 within particular venues may be an additional threat due to the misalignment of
3801 search query keywords to intended papers of inclusion. Thus, our conclusions are
3802 only applicable to the information we were able to extract and summarise, given the
3803 primary sources selected.

3804 While we have investigated the application of this taxonomy using a user study
3805 (Section 7.5.1), we would like to explore an observational study of developers
3806 to assess how improved and non-improved API documentation impacts developer
3807 productivity. The outcome of this work can help design a follow-up experiment,
3808 consisting of a comparative controlled study [296] that capture firsthand behaviours
3809 and interactions toward how software engineers approach using a CVS with and
3810 without our taxonomy applied. This can be achieved by providing ‘mock’ improved
3811 documentation with the suggested improvements included in this work. Such an ex-
3812 periment could recruit a sample of developers of varying experience (from beginner
3813 programmer to principal engineer) to complete a certain number of tasks under an
3814 observational, comparative controlled study, half of which will (a) develop using
3815 the improved ‘mock’ documentation, and the other half will (b) develop with the
3816 *as-is/existing* documentation. From this, we can compare if the framework makes
3817 improvements by capturing metrics and recording the observational sessions for
3818 qualitative analysis. Visual modelling can be adopted to analyse the qualitative data
3819 using matrices [92], maps and networks [293] as these help illustrate any causal, tem-

3820 poral or contextual relationships that may exist to map out the developer's mindset
3821 and difference in approaching the two sets of designs of the same tasks.

3822 7.8 Conclusions & Future Work

3823 A good API document should facilitate a developer's productivity, and is therefore
3824 associated to the quality of software produced; improving the quality of the docu-
3825 mentation of third-party APIs improves the quality of dependent software. However,
3826 there does not yet exist a consolidated taxonomy of key recommendations proposed
3827 by literature, and—more importantly—it is useful to know if what developers need
3828 *in-practice* differs to what documentation artefacts are anticipated by literature.
3829 Moreover, there has been little work on mapping the research produced in this space
3830 against the techniques used to arrive at the recommendations.

3831 This study priorities which aspects of API documentation knowledge is both (i)
3832 suggested by literature, and (ii) is demanded *most* by developers. We conduct a
3833 SMS from a pool of 4,501 studies and identify 21 seminal studies. From this, we
3834 synthesise a taxonomy of the various documentation aspects that should improve
3835 API documentation quality. Furthermore, we also capture the most commonly used
3836 analysis techniques used in the academic literature. We then validate our taxonomy
3837 against developers to assess its efficacy with practitioners, and conduct a heuristic
3838 evaluation against to three popular CVSs. We offer 12 detailed suggested improve-
3839 ments where these services currently have weaknesses, and where specifically they
3840 may be able to improve their documentation.

3841 Future extensions of our work may involve a restricted systematic literature
3842 review in API documentation artefacts, and many suggestions are further detailed
3843 in Section 7.7. Further, a review into the techniques of these primary studies may
3844 extend the mapping we conducted in this work, by evaluating the the effectiveness of
3845 the various approaches used in each study and assessing these against the proposed
3846 conclusions of each study.

3847 The findings of our work provides a solid baseline for improving the documen-
3848 tation of non-deterministic software, such as CVSs. While our aim is to eventually
3849 improve the quality of API documentation, the ultimate goal is to improve the soft-
3850 ware engineer's experience of non-deterministic IWSs. We hope the guidelines from
3851 this extensive study help both software developers and API providers alike by using
3852 our taxonomy as a go-to checklist for what should be considered in documenting any
3853 API.

CHAPTER 8

3854

3855

3856 Using a Facade Pattern to combine Computer Vision Services[†]

3857

3858 **Abstract** Intelligent computer vision services, such as Google Cloud Vision or Amazon
3859 Rekognition, are becoming evermore pervasive and easily accessible to developers to build
3860 applications. Because of the stochastic nature that ML entails and disparate datasets used in
3861 their training, the outputs from different computer vision services varies with time, resulting
3862 in low reliability—for some cases—when compared against each other. Merging multiple
3863 unreliable API responses from multiple vendors may increase the reliability of the overall
3864 response, and thus the reliability of the intelligent end-product. We introduce a novel
3865 methodology—inspired by the proportional representation used in electoral systems—to
3866 merge outputs of different intelligent computer vision API provided by multiple vendors.
3867 Experiments show that our method outperforms both naive merge methods and traditional
3868 proportional representation methods by 0.015 F-measure.

3869 8.1 Introduction

3870 With the introduction of intelligent web services (IWSs) that make machine learning
3871 (ML) more accessible to developers [274, 337], we have seen a large growth of
3872 intelligent applications dependent on such services [59, 126]. For example, consider
3873 the advances made in computer vision, where objects are localised within an image
3874 and labelled with associated categories. Cloud-based computer vision services
3875 (CVSs)—e.g., [363, 376, 384, 388, 397, 398, 402, 450]—are a subset of IWSs.
3876 They utilise ML techniques to achieve image recognition via a remote black-box
3877 approach, thereby reducing the overhead for application developers to understand
3878 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.

3879 and training of the machine-learnt algorithms is offloaded to the cloud, developers
3880 simply send RESTful API requests to do the recognition. There are, however, inherit
3881 differences and drawbacks between traditional web services and IWSs, which we
3882 describe with the motivating scenario below.

3883 8.1.1 Motivating Scenario: Intelligent vs Traditional Web Services

3884 An application developer, Tom, wishes to develop a social media Android and iOS
3885 app that catalogues photos of him and his friends, common objects in the photo,
3886 and generates brief descriptions in the photo (e.g., all photos with his husky dog,
3887 all photos on a sunny day etc.). Tom comes from a typical software engineering
3888 background with little knowledge of computer vision and its underlying concepts.
3889 He knows that intelligent computer vision web APIs are far more accessible than
3890 building a computer vision engine from scratch, and opts for building his app using
3891 these cloud services instead.

3892 Based on his experiences using similar cloud services, Tom would expect consistency
3893 of the results from the same API and different APIs that provide the same (or
3894 similar) functionality. As an analogy, when Tom writes the Java substring method
3895 "`doggy".substring(0, 2)`", he expects it to be the same result as the Swift equivalent
3896 "`"doggy".prefix(3)`". Each and every time he interacts with the substring
3897 method using either API, he gets "dog" as the response. This is because Tom is
3898 used to deterministic, rule-driven APIs that drive the implementation behind the
3899 substring method.

3900 Tom's deterministic mindset results in three key differentials between a traditional
3901 web services and an IWS:

3902 **(1) Given similar input, results differ between similar IWSs.** When Tom
3903 interacts with the API of an IWS, he is not aware that each API provider trains
3904 their own, unique ML model, both with disparate methods and datasets. These
3905 IWSs are, therefore, nondeterministic and data-driven; input images—even
3906 if they contain the same conceptual objects—often output different results.
3907 Contrast this to the substring method of traditional APIs; regardless of what
3908 programming language or string library is used, the same response is expected
3909 by developers.

3910 **(2) Intelligent responses are not certain.** When Tom interprets the response
3911 object of an IWS, he finds that there is a ‘confidence’ value or ‘score’. This
3912 is because the ML models that power IWSs are inherently probabilistic and
3913 stochastic; any insight they produce is purely statistical and associational [258].
3914 Unlike the substring example, where the rule-driven implementation provides
3915 certainty to the results, this is not guaranteed for IWSs. For example, a picture
3916 of a husky breed of dog is misclassified as a wolf. This could be due to
3917 adversarial examples [317] that ‘trick’ the model into misclassifying images
3918 when they are fully decipherable to humans. It is well-studied that such
3919 adversarial examples exist in the real world unintentionally [106, 189, 261].

3920 **(3) Intelligent APIs evolve over time.** Tom may find that responses to processing
3921 an image may change over time; the labels he processes in testing may evolve

3922 and therefore differ to when in production. In traditional web services, evo-
3923 lution in responses is slower, generally well-communicated, and usually rare
3924 (Tom would always expect "dog" to be returned in the substring example).
3925 This has many implications on software systems that depend on these APIs,
3926 such as confidence in the output and portability of the solution. Currently, if
3927 Tom switches from one API provider to another, or if he doesn't regularly test
3928 his app in production, he may begin to see a very different set of labels and
3929 confidence levels.

3930 8.1.2 Research Motivation

3931 These drawbacks bring difficulties to the intended API users like Tom. We identify a
3932 gap in the software engineering literature regarding such drawbacks, including: lack
3933 of best practices in using IWSs; assessing and improving the reliability of APIs for
3934 their use in end-products; evaluating which API is suitable for different developer
3935 and application needs; and how to mitigate risk associated with these APIs. We
3936 focus on improving reliability of CVSs for use in end-products. The key research
3937 questions in this paper are:

3938 **RQ1:** Is it possible to improve reliability by merging multiple CVS results?

3939 **RQ2:** Are there better algorithms for merging these results than currently in
3940 use?

3941 Previous attempts at overcoming low reliability include triple-modular redun-
3942 dancy [207]. This method uses three modules and decides output using majority
3943 rule. However, in CVSs, it is difficult to apply majority rule: these APIs respond with
3944 a list of labels and corresponding scores. Moreover, disparate APIs ordinarily output
3945 different results. These differences make it hard to apply majority rule because the
3946 type of outputs are complex and disparate APIs output different results for the same
3947 input. Merging search results is another technique to improve reliability [304]. It
3948 normalises scores of different databases using a centralised sample database. Nor-
3949 malising scores makes it possible to merge search results into a single ranked list.
3950 However, search responses are disjoint, whereas they are not in the context of most
3951 CVSs.

3952 In this paper, we introduce a novel method to merge responses of CVSs, using
3953 image recognition APIs endpoints as our motivating example. Section 8.2 describes
3954 naive merging methods and requirements. Section 8.3 gives insights into the struc-
3955 ture of labels. Section 8.4 introduces our method of merging computer vision labels.
3956 Section 8.5 compares precision and recall for each method. Section 8.6 presents
3957 conclusions and future work.

3958 8.2 Merging API Responses

3959 Image recognition APIs have similar interfaces: they receive a single input (image)
3960 and respond with a list of labels and associated confidence scores. Similarly, other
3961 supervised-AI-based APIs do the same (e.g., detecting emotions from text and
3962 natural language processing [399, 451]). It is difficult to apply majority rule on such

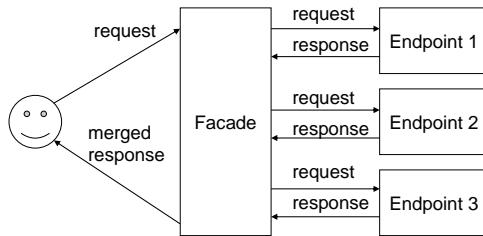


Figure 8.1: The user sends a request to the facade; this request is propagated to the relevant APIs. Responses are merged by the facade and returned back to the user.

3963 disparate, complex outputs. While the outputs by *multiple* AI-based API endpoints
 3964 is different and complex, the general format of the output is the same: a list of labels
 3965 and associated scores.

3966 8.2.1 API Facade Pattern

3967 To merge responses from multiple APIs, we introduce the notion of an API facade.
 3968 It is similar to a metasearch engine, but differs in their external endpoints. The
 3969 facade accepts the input from one API endpoint (the facade endpoint), propagates
 3970 that input to all user-registered concrete (external) API endpoints simultaneously,
 3971 then ‘merges’ outputs from these concrete endpoints before sending this merged
 3972 response to the API user. We demonstrate this process in Figure 8.1.

3973 Although the model introduces more time and cost overhead, both can be mitigated
 3974 by caching results. On the other hand, the facade pattern provides the following
 3975 benefits:

- 3976 • **Easy to modify:** It requires only small modifications to applications, e.g.,
 3977 changing each concrete endpoint URL.
- 3978 • **Easy to customise:** It merges results from disparate and concrete APIs ac-
 3979 cording to the user’s preference.
- 3980 • **Improves reliability:** It enhances reliability of the overall returned result by
 3981 merging results from different endpoints.

3982 8.2.2 Merge Operations

3983 The API facade is applicable to many use cases. However, this paper focuses on
 3984 APIs that output a list of labels and scores, as is the case for CVSs. Merge operations
 3985 involve the mapping of multiple lists and associated scores, produced by multiple
 3986 APIs, to just one list. For instance, a CVS receives a bowl of fruit as the input image
 3987 and outputs the following:

3988 `[[‘apple’, 0.9], [‘banana’, 0.8]]`

3989 where the first item is the label and the second item is the score. Similarly, another
 3990 computer vision API outputs the following for the same image:

3991 `[[‘apple’, 0.7], [‘cherry’, 0.8]].`

3992 Merge operations can, therefore, merge these two responses into just one response.
3993 Naive ways of merging results could make use of *max*, *min*, and *average* operations
3994 on the confidence scores. For example, *max* merges results to:

3995 $\text{[['apple', 0.9], ['banana', 0.8], ['cherry', 0.8]]};$

3996 *min* merges results to:

3997 $\text{[['apple', 0.7]]};$

3998 and *average* merges results to:

3999 $\text{[['apple', 0.8], ['banana', 0.4], ['cherry', 0.4]]}.$

4000 However, as the object's labels in each result are natural language, the operations
4001 do not exploit the label's semantics when conducting label merging. To improve
4002 the quality of the merged results, we consider the ontologies of these labels, as we
4003 describe below.

4004 8.2.3 Merging Operators for Labels

4005 Merge operations on labels are *n*-ary operations that map R^n to R , where $R_i =$
4006 $\{(l_{ij}, s_{ij})\}$ is a response from endpoint i and contains pairs of labels (l_{ij}) and scores
4007 (s_{ij}). Merge operations on labels have the following properties:

- 4008 • *identity* defines that merging a single response should output same response
4009 (i.e., $R = \text{merge}(R)$ is always true);
- 4010 • *commutativity* defines that the order of operands should not change the result
4011 (i.e., $\text{merge}(R_1, R_2) = \text{merge}(R_2, R_1)$ is always true);
- 4012 • *reflexivity* defines that merging multiple same responses should output same
4013 response (i.e., $R = \text{merge}(R, R)$ is always true); and,
- 4014 • *additivity* defines that, for a specific label, the merged response should have
4015 higher or equal score for the label if a concrete endpoint has a higher score.
4016 Let $R = \text{merge}(R_1, R_2)$ and $R' = \text{merge}(R'_1, R_2)$ be merged responses. R_1 and
4017 R'_1 are same, except R'_1 has a higher score for label l_x than R_1 . The additive
4018 score property requires that R' score for l_x should be greater than or equal to
4019 R score for l_x .

4020 The *max*, *min*, and *average* operations in Section 8.2.2 follow each of these rules
4021 as all operations calculate the score by applying these operations on each score.

4022 8.3 Graph of Labels

4023 CSVs typically return lists of labels and their associated scores. In most cases, the
4024 label can be a singular word (e.g., ‘husky’) or multiple words (e.g., ‘dog breed’).
4025 Lexical databases, such as WordNet [227], can therefore be used to describe the
4026 ontology behind these labels’ meanings. Figure 8.2 is an example of a graph of

Table 8.1: 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

4027 labels and synsets. A synset is a grouped set of synonyms for a word. In this image,
 4028 we consider two fictional endpoints, endpoints 1–2. We label red nodes as labels
 4029 from endpoint 1, yellow nodes as labels from endpoint 2, and blue nodes as synsets
 4030 for the associated labels from both endpoints. As actual graphs are usually more
 4031 complex, Figure 8.2 is a simplified graph to illustrate the usage of associating labels
 4032 from two concrete sources to synsets.

4033 8.3.1 Labels and synsets

4034 The number of labels depends on input images and concrete API endpoints used.
 4035 Table 8.1 and Figure 8.3 show how many labels are returned, on average per image,
 4036 from Google Cloud Vision [388], Amazon Rekognition [363] and Azure Computer
 4037 Vision [402] image recognition APIs. These statistics were calculated using 1,000
 4038 images from Open Images Dataset V4 [390] Image-Level Labels set.

4039 Labels from Amazon and Microsoft tend to have corresponding synsets, and
 4040 therefore these endpoints return common words that are found in WordNet. On the
 4041 other hand, Google’s labels have less corresponding synsets: for example, labels
 4042 without corresponding synsets are car models and dog breeds.¹

4043 8.3.2 Connected Components

4044 A connected component (CC) is a subgraph in which there are paths between any
 4045 two nodes. In graphs of labels and synsets, CCs are clusters of labels and synsets
 4046 with similar semantic meaning. For instance, there are two CCs in Figure 8.2. CC 1
 4047 in Figure 8.2 has ‘beverage’, ‘dessert’, ‘chocolate’, ‘hot chocolate’,
 4048 ‘drink’, and ‘food’ labels from the red first endpoint and ‘coffee’, ‘hot
 4049 chocolate’, ‘drink’, ‘caffeine’, and ‘tea’ labels from the yellow second
 4050 endpoint. Therefore, these labels are related to ‘drink’. On the other hand, CC 2
 4051 in Figure 8.2 has ‘cup’ and ‘coffee cup’ labels from the first red endpoint and
 4052 ‘cup’, ‘coffee cup’, and ‘tableware’ labels from the yellow second endpoint.
 4053 These labels are, therefore, related to ‘cup’.

4054 Figure 8.4 shows a distribution of number of CCs for the 1,000-image label
 4055 detections on Amazon Rekognition, Google Cloud Vision, and Azure Computer
 4056 Vision APIs. The average number of CCs is 9.36 ± 3.49 . The smaller number of
 4057 CCs means that most of labels have similar meanings, while a larger value means
 4058 that the labels are more disparate.

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

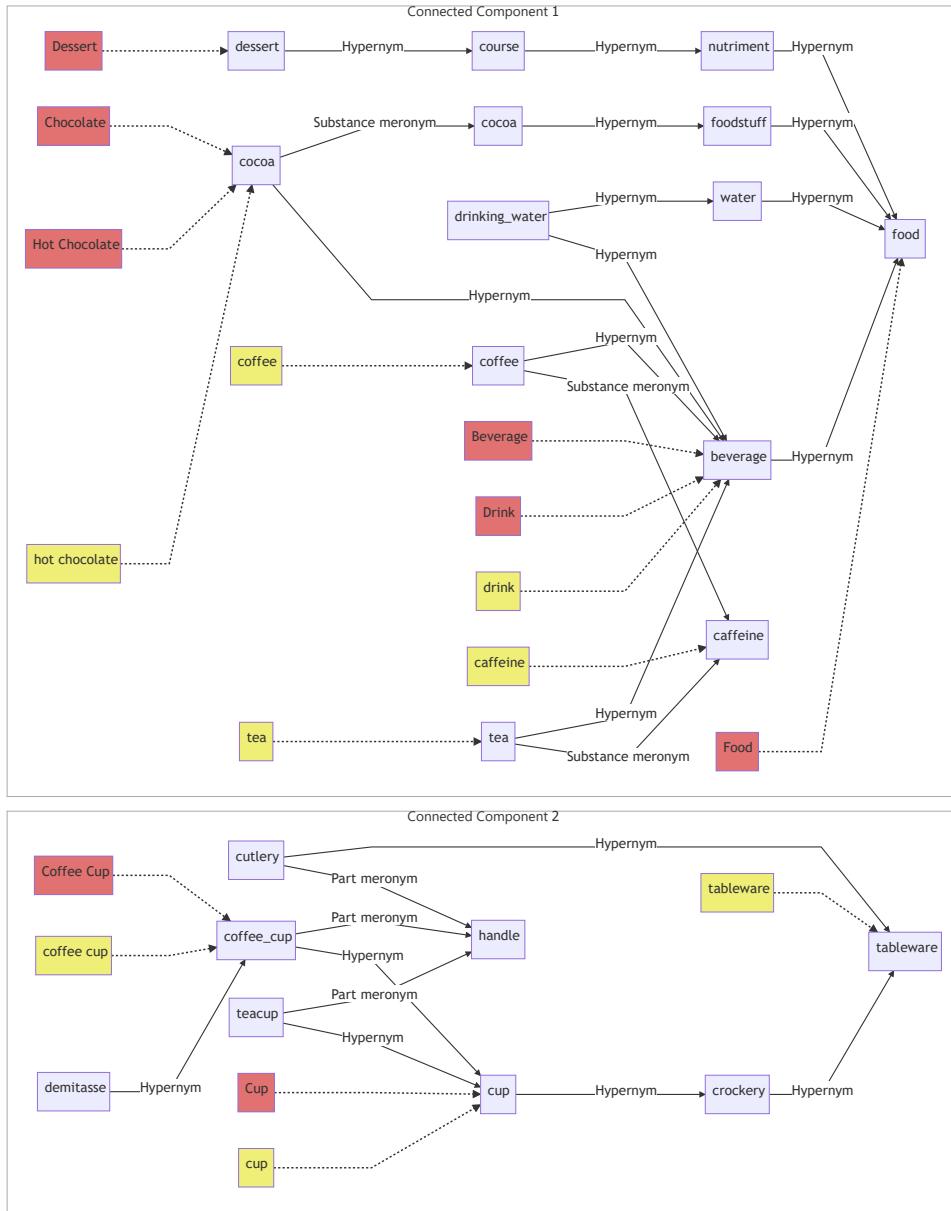


Figure 8.2: Graph of labels from two concrete endpoints (red and yellow) and their associated synsets related to both words (blue).

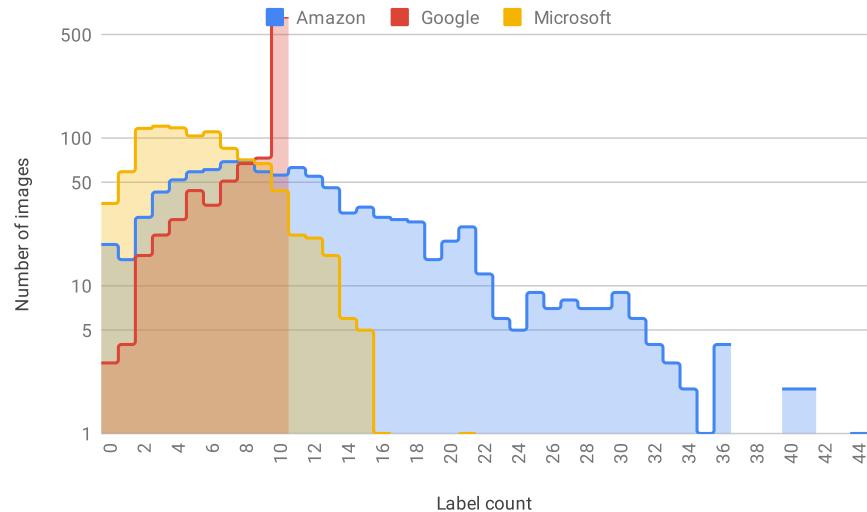


Figure 8.3: Number of labels responded from our input dataset to three concrete APIs assessed.

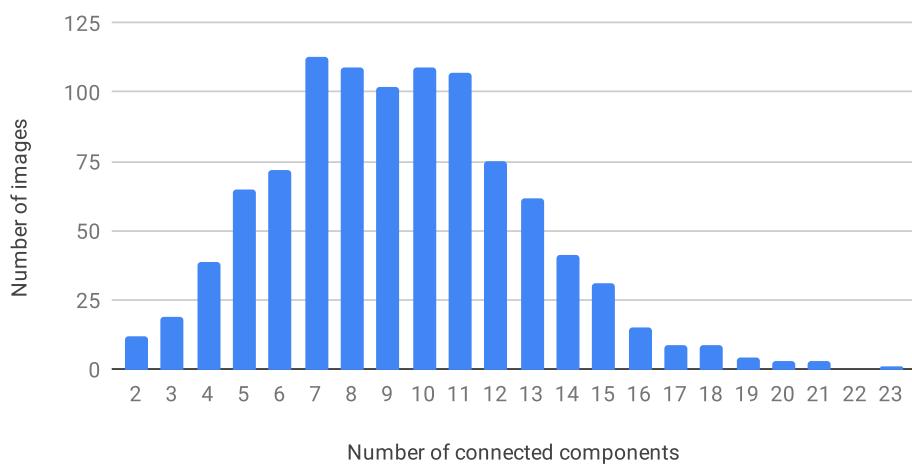


Figure 8.4: Number of connected components compared to the number of images.

4059 8.4 API Results Merging Algorithm

4060 Our proposed algorithm to merge labels consists of four parts: (1) mapping labels to
 4061 synsets, (2) deciding the total number of labels, (3) allocating the number of labels
 4062 to CCs, and (4) selecting labels from CCs.

4063 8.4.1 Mapping Labels to Synsets

4064 Labels returned in CVS responses are words (in natural language) that do not always
 4065 identify their intended meanings. For instance, a label *orange* may represent the
 4066 fruit, the colour, or the name of the longest river in South Africa. To identify the
 4067 actual meanings behind a label, our facade enumerates all synsets corresponding to
 4068 labels. It then finds the most likely synsets for labels by traversing WordNet links.
 4069 For instance, if an API endpoint outputs the ‘*orange*’ and ‘*lemon*’ labels, the
 4070 facade regards ‘*orange*’ as a related synset word of ‘*fruit*’. If an API endpoint
 4071 outputs ‘*orange*’ and ‘*water*’ labels, the facade regards ‘*orange*’ as a ‘*river*’.

4072 8.4.2 Deciding Total Number of Labels

4073 The number of labels in responses from endpoints vary as described in Section 8.3.1.
 4074 The facade decides the number of merged labels using the numbers of labels from
 4075 each endpoint. We formulate the following equation to calculate the number of
 4076 labels:

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

4077 where $|R|$ is number of labels and scores in response, and n is number of endpoints.
 4078 In case of naive operations in Section 8.2.2, the following is true:

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

4079 The proposal uses $\lfloor \sum_i |R_i| / n \rfloor$ to conform to the necessary condition described in
 4080 Section 8.4.3.

4081 8.4.3 Allocating Number of Labels to Connected Components

4082 The graph of labels and synsets is then divided into several CCs. The facade decides
 4083 how many labels are allocated for each CC. For example, in Figure 8.5, there are
 4084 three CCs, where square-shaped nodes are labels in responses from endpoints. Text
 4085 within these label nodes describe which endpoint outputs the label and score, for
 4086 instance, “L-1a, 0.9” is label *a* from endpoint *L* with a score 0.9. Circle-shaped nodes
 4087 represent synsets, where the edges between the label and synset nodes indicate the
 4088 relationships between them. Edges between synsets are links in WordNet.

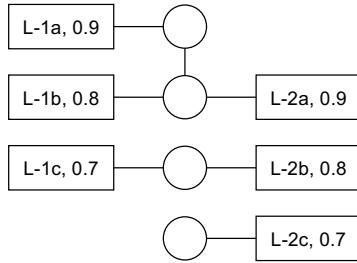


Figure 8.5: Allocation to connected components.

4089 Allegorically, allocating the number of labels to CCs is similar to proportional
 4090 representation in a political voting system, where CCs are the political parties and
 4091 labels are the votes to a party. Several allocation algorithms are introduced in
 4092 proportional representation, for instance, the D'Hondt and Hare-Niemeyer methods
 4093 [239]. However, there are differences from proportional representation in the polit-
 4094 ical context. For label merging, labels have scores and origin endpoints and such
 4095 information may improve the allocation algorithm. For instance, CCs supported
 4096 with more endpoints should have a higher allocation than CCs with fewer endpoints,
 4097 and CCs with higher scores should have a higher allocation than CCs with lower
 4098 scores. We introduce an algorithm to allocate the number of labels to CCs. This
 4099 allocates more to a CC with more supporting endpoints and higher scores. The steps
 4100 of the algorithm are:

- 4101 **Step I.** Sort scores separately for each endpoint.
- 4102 **Step II.** If all CCs have an empty score array or more, remove one, and go to Step
 4103 II.
- 4104 **Step III.** Select the highest score for each endpoint and calculate product of highest
 4105 scores.
- 4106 **Step IV.** A CC with the highest product score receives an allocation. This CC
 4107 removes every first element from the score array.
- 4108 **Step V.** If the requested number of allocations is complete, then stop allocation.
 4109 Otherwise, go to Step II.

4110 Tables 8.2 to 8.5 are examples of allocation iterations. In Table 8.2, the facade
 4111 sorts scores separately for each endpoint. For instance, the first CC in Figure 8.5
 4112 has scores of 0.9 and 0.8 from endpoint 1 and 0.9 from endpoint 2. All CCs have a
 4113 non-empty score array or more, so the facade skips Step II. The facade then picks
 4114 the highest scores for each endpoint and CC. CC 1 has the largest product of highest
 4115 scores and receives an allocation. In Table 8.3, the first CC removes every first score
 4116 in its array as it received an allocation in Table 8.2. In this iteration, the second CC
 4117 has largest product of scores and receives an allocation. In Table 8.4, the second CC
 4118 removes every first score in its array. At Step II, all the three CCs have an empty
 4119 array. The facade removes one empty array from each CC. In Table 8.5, the first CC
 4120 receives an allocation. The algorithm is applicable if total number of allocation is

Table 8.2: 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 8.4: Allocation iteration 3.

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

Table 8.3: 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 8.5: 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

4121 less than or equal to $\max_i(|R_i|)$ as scores are removed in Step II. The condition is a
4122 necessary condition.

4123 8.4.4 Selecting Labels from Connected Components

4124 For each CC, the facade applies the *average* operator from Section 8.2.2 and takes
4125 labels with n -highest scores up to allocation, as per Section 8.4.3.

4126 8.4.5 Conformance to properties

4127 Section 8.2.3 defines four properties: identity, commutativity, reflexivity, and additivity.
4128 Our proposed method conforms to these properties:

- 4129 • *identity*: the method outputs same result if there is one response;
- 4130 • *commutativity*: the method does not care about ordering of operands;
- 4131 • *reflexivity*: the allocations to CCs are same to number of labels in CCs; and
- 4132 • *additivity*: increases in score increases or does not change the allocation to
4133 the corresponding CC.

4134 8.5 Evaluation

4135 8.5.1 Evaluation Method

4136 To evaluate the merge methods, we merged CVS results from three representative
4137 image analysis API endpoints and compared these merged results against human-
4138 verified labels. Images and human-verified labels are sourced from 1,000 randomly-
4139 sampled images from the Open Images Dataset V4 [390] Image-Level Labels test
4140 set.

4141 The first three rows in Table 8.7 are the evaluation of original responses from
4142 each API endpoint. Precision, recall, and F-measure in Table 8.7 do not reflect
4143 actual values: for instance, it appears that Google performs best at first glance, but
4144 this is mainly because Google’s labels are similar to that of the Open Images label
4145 set.

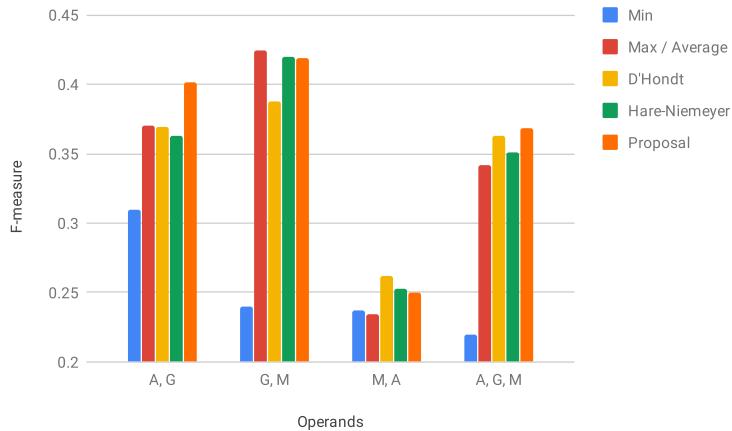


Figure 8.6: F-measure comparison.

4146 The Open Images Dataset uses 19,995 classes for labelling. The human-verified
 4147 labels for the 1,000 images contain 8,878 of these classes. Table 8.6 shows the
 4148 correspondence between each service's labels and the Open Images Dataset classes.
 4149 For instance, Amazon Rekognition outputs 11,416 labels in total for 1,000 images.
 4150 There are 1,409 unique labels in 11,416 labels. 1,111 labels out of 1,409 can be
 4151 found in Open Images Dataset classes. Rekognition's labels matches to Open Images
 4152 Dataset classes at 78.9% ratio, while Google has an outstanding matched percentage
 4153 of 94.1%. This high match is likely due to Google providing both Google Cloud
 4154 Vision and the Open Images Dataset—it is likely that they are trained on the same
 4155 data and labels. An endpoint with higher matched percentage has a more similar
 4156 label set to the Open Images Dataset classes. However, a higher matched percentage
 4157 does not mean imply *better quality* of an API endpoint; it will increase apparent
 4158 precision, recall, and F-measure only.

4159 The true and false positive (TP/FP) label averages and the TP/FP ratio is shown
 4160 in Table 8.7. Where the TP/FP ratio is larger, the scores are more reliable, however
 4161 it is possible to increase the TP/FP ratio by adding more false labels with low scores.
 4162 On the other hand, it is impossible to increase F-measure intentionally, because
 4163 increasing precision will decrease recall, and vice versa. Hence, the importance of
 4164 the F-measure statistic is critical for our analysis.

4165 Let R_A , R_G , and R_M be responses from Amazon Rekognition, Google Cloud
 4166 Vision, and Microsoft's Azure Computer Vision, respectively. There are four sets of
 4167 operands, i.e., (R_A, R_G) , (R_G, R_M) , (R_M, R_A) , and (R_A, R_G, R_M) . Table 8.7 shows the
 4168 evaluation of each operands set, Table 8.8 shows the averages of the four operands
 4169 sets, and Figure 8.6 shows the comparison of F-measure for each methods.

4170 8.5.2 Naive Operators

4171 Results of *min*, *max*, and *average* operators are shown in Tables 8.7 and 8.8 and Fig-
 4172 ure 8.6. The *min* operator is similar to *union* operator of set operation, and outputs
 4173 all labels of operands. The precision of the *min* operator is always greater than any

Table 8.6: 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 8.7: 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 8.8: 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

4174 precision of operands, and the recall is always lesser than any precision of operands.
4175 *Max* and *average* operators are similar to *intersection* operator of set operations.
4176 Both operators output intersection of labels of operands and there is no clear relation
4177 to the precision and recall of operands. Since both operators have the same preci-
4178 sion, recall, and F-measure, Figure 8.6 groups them into one. The *average* operator
4179 performs well on the TP/FP ratio, where most of the same labels from multiple
4180 endpoints are TPs. In many cases of the four operand sets, all naive operators'
4181 F-measures are between F-measures of operands. None of naive operators therefore
4182 improve results by merging responses from multiple endpoints.

4183 8.5.3 Traditional Proportional Representation Operators

4184 There are many existing allocation algorithms in proportional representation, e.g.,
4185 the Niemeyer and Niemeyer method [239]. These methods may be replacements of
4186 those in Section 8.4.3. Other steps, i.e., Sections 8.4.1, 8.4.2 and 8.4.4, are the same
4187 as for our proposed technique. Tables 8.7 and 8.8 and Figure 8.6 show the result of
4188 these traditional proportional representation algorithms. Averages of F-measures by
4189 traditional proportional representation operators are almost equal to that of the *max*
4190 and *average* operators. It is worth noting that merging *M* and *A* responses results in
4191 a better F-measure than each F-measure of *M* and *A* individually. As these are not
4192 biased to human-verified labels, situations in the real-world usage should, therefore,
4193 be similar to the case of *M* and *A*. Hence, RQ1 is true.

4194 8.5.4 New Proposed Label Merge Technique

4195 As shown in Table 8.8, our proposed new method performs best in F-measure.
4196 Instead, the TP/FP ratio is less than *average*, the D'Hondt method, and Hare-
4197 Niemeyer method. As described in Section 8.5.1, we argue that F-measure is a
4198 more important measure than the TP/FP ratio (in this case). Therefore, RQ2 is
4199 true. Shown in Table 8.7, our proposed new method improves the results when
4200 merging *M* and *A* in non-biased endpoints. It is similar to traditional proportional
4201 representation operators, but does not perform as well. However, it performs better
4202 on other operand sets, and performs best overall as shown in Figure 8.6.

4203 8.5.5 Performance

4204 We used AWS EC2 m5.large instance (2 vCPUs, 2.5 GHz Intel Xeon, 8 GiB RAM);
4205 Amazon Linux 2 AMI (HVM), SSD Volume Type; Node.js 8.12.0. It takes 0.370
4206 seconds to merge responses from three endpoints. Computational complexity of the
4207 algorithm in Section 8.4.3 is $O(n^2)$, where n is total number of labels in responses.
4208 (The estimation assumes that the number of endpoints is a constant.) Complexity
4209 of Step I in Section 8.4.3 is $O(n \log n)$, as the worst case is that all n labels are from
4210 one single endpoint and all n labels are in one CC. Complexity of Step II to Step V
4211 is $O(n^2)$, as the number of CCs is less than or equal to n and number of iterations
4212 are less than or equal to n . As Table 8.1 shows, the averaged total number of three
4213 endpoints is 25.58. Most of time for merging is consumed by looking up WordNet

4214 synsets (Section 8.4.1). The API facade calls each APIs on actual endpoints in
4215 parallel. It takes about 5 seconds, which is much longer than 0.370 seconds taken
4216 for the merging of responses.

4217 8.6 Conclusions and Future Work

4218 In this paper, we propose a method to merge responses from CVSSs. Our method
4219 merges API responses better than naive operators and other proportional represen-
4220 tation methods (i.e., D’Hondt and Hare-Niemeyer). The average of F-measure of
4221 our method marks 0.360; the next best method, Hare-Niemeyer, marks 0.347. Our
4222 method and other proportional representation methods are able to improve the F-
4223 measure from original responses in some cases. Merging non-biased responses
4224 results in an F-measure of 0.250, while original responses have an F-measure be-
4225 tween 0.246 and 0.242. Therefore, users can improve their applications’ precision
4226 with small modification, i.e., by switching from a singular URL endpoint to a facade-
4227 based architecture. The performance impact by applying facades is small, because
4228 overhead in facades is much smaller than API invocation. Our proposal method
4229 conforms identity, commutativity, reflexivity, and additivity properties and these
4230 properties are advisable for integrating multiple responses.

4231 Our idea of a proportional representation approach can be applied to other IWSs.
4232 If the response of such a service is list consisting of an entity and score, and if there is a
4233 way to group entities, a proposal algorithm can be applied. The opposite approach is
4234 to improve results by inferring labels. Our current approach picks some of the labels
4235 returned by endpoints. IWSs are not only based on supervised ML—thus to cover a
4236 wide range of IWSs, it is necessary to classify and analyse each APIs and establish
4237 a method to improve results by merging. Currently graph structures of labels and
4238 synsets (Figure 8.2) are not considered when merging results. Propagating scores
4239 from labels could be used, losing the additivity property but improving results for
4240 users. There are many ways to propagate scores. For instance, setting propagation
4241 factors for each link type would improve merging and could be customised for users’
4242 preferences. It would be possible to generate an API facade automatically. APIs
4243 with the same functionality have same or similar signatures. Machine-readable API
4244 documentation, for instance, OpenAPI Specification, could help a generator to build
4245 an API facade.

CHAPTER 9

4246

4247

4248

Threshy: Supporting Safe Usage of Intelligent Web Services[†]

4249

4250 **Abstract** Increased popularity of ‘intelligent’ web services provides end-users with machine-
4251 learnt functionality at little effort to developers. However, these services require a decision
4252 threshold to be set which is dependent on problem-specific data. Developers lack a systematic
4253 approach for evaluating intelligent services and existing evaluation tools are predominantly
4254 targeted at data scientists for pre-development evaluation. This paper presents a workflow
4255 and supporting tool, Threshy, to help *software developers* select a decision threshold suited
4256 to their problem domain. Threshy is designed for tuning the confidence scores returned by
4257 intelligent web services and does not deal with hyper-parameter optimisation used in ML
4258 models. Additionally, it considers the financial impacts of false positives. Unlike existing
4259 tools, Threshy is designed to operate in multiple workflows including pre-development, pre-
4260 release, and support. Threshold configuration files exported by Threshy can be integrated
4261 into client applications and monitoring infrastructure. Demo: <https://bit.ly/2YKeYhE>.

9.1 Introduction

4262 Machine learning algorithm adoption is increasing in modern software. End users
4263 routinely benefit from machine-learnt functionality through personalised recom-
4264 mendations [76], voice-user interfaces [234], and intelligent digital assistants [46]. The
4265 easy accessibility and availability of intelligent web services (IWSs)¹ is contribut-
4266 ing to their adoption. These IWSs simplify the development of machine learning
4267 solutions as they (i) do not require specialised machine learning expertise to build
4268 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/>).

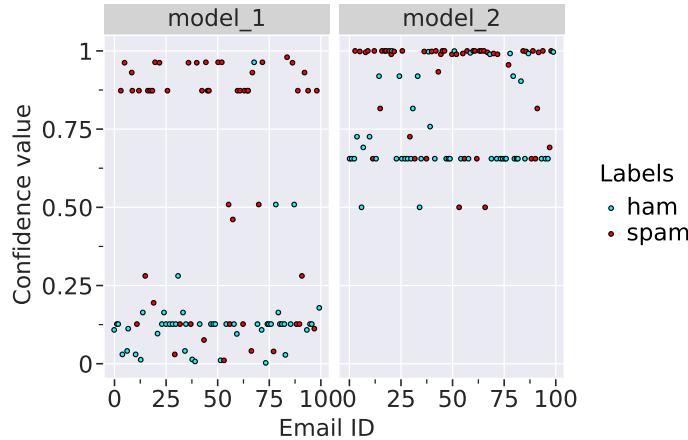


Figure 9.1: Predictions for 100 emails from two spam classifiers. Decision thresholds are classifier-dependent: a single threshold for both classifiers is *not* appropriate as ham emails are clustered at 0.12 (model_1) and at 0.65 (model_2). Developers must evaluate performance for *both* thresholds.

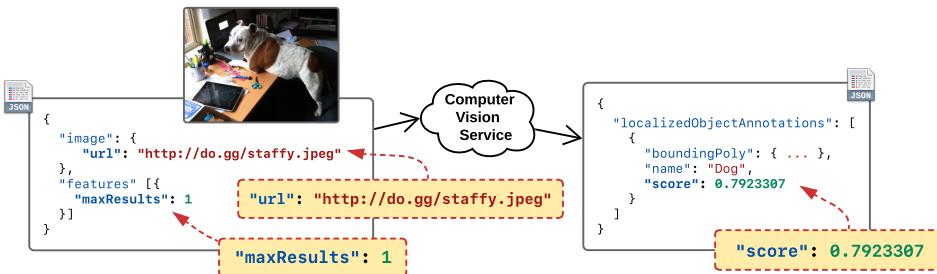


Figure 9.2: Request and response for an intelligent computer vision web service with only three configuration parameters: the image’s `url`, `maxResults` and `score`.

learning [11, 295], 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 [81]. 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 10.5, 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

4286 parameters of the algorithm for training a model. In this case, developers have no
4287 insight into which hyperparameters are used or the algorithm selected and cannot
4288 tune the trained model. Thus an evaluation procedure must be followed as a part of
4289 using an intelligent service for an application.

4290 A typical evaluation process would involve a test data set (curated by the devel-
4291 opers using the intelligent service) that is used to determine an appropriate threshold.
4292 Choice of a decision threshold is a critical element of the evaluation procedure [138].
4293 This is especially true for classification problems such as detecting if an image con-
4294 tains cancer or identifying all of the topics in a document. Simple approaches
4295 to selecting a threshold are often insufficient, as highlighted in Google’s machine
4296 learning course: “*It is tempting to assume that [a] classification threshold should
4297 always be 0.5, but thresholds are problem-dependent, and are therefore values that
4298 you must tune.*”² As an example consider the predictions from two email spam
4299 classifiers shown in Figure 9.1. The predicted safe emails, ‘ham’, are in two separate
4300 clusters (a simple threshold set to approx. 0.2 for model 1 and 0.65 for model 2),
4301 indicating that different decision thresholds may be required depending on the clas-
4302 sifier. Also note that some emails have been misclassified; how many depends on
4303 the choice of decision threshold. An appropriate threshold considers factors outside
4304 algorithmic performance, such as financial cost and impact of wrong decisions. To
4305 select an appropriate decision threshold, developers using intelligent services need
4306 approaches to reason about and consider trade-offs between competing *cost fac-
4307 tors*. These include impact, financial costs, and maintenance implications. Without
4308 considering these trade-offs, sub-optimal decision thresholds will be selected.

4309 The standard approach for tuning thresholds in classification problems involve
4310 making trade-offs between the number of false positives and false negatives using
4311 the receiver operating characteristic (ROC) curve. However, developers (i) need
4312 to realise that this trade-off between false positives and false negatives is a data
4313 dependent optimisation process [294], (ii) often need to develop custom scripts
4314 and follow a trial-and-error based approach to determine a threshold, (iii) must
4315 have appropriate statistical training and expertise, and (iv) be aware that multi-
4316 label classification require more complex optimisation methods when setting label
4317 specific costs. However, current intelligent services do not sufficiently guide or
4318 support software engineers through the evaluation process, nor do they make this
4319 need clear in the documentation.

4320 In this paper we present **Threshy**³, a tool to assist developers in selecting de-
4321 cision thresholds when using intelligent services. The motivation for developing
4322 Threshy arose from our consultancy work with industry. Unlike existing tooling
4323 (see Section 9.4), **Threshy serves as a means to up-skill and educate software en-
4324 gineers in selecting machine-learnt decision thresholds**, for example, on aspects
4325 such as confusion matrices. We re-iterate that the end-users of Threshy are software
4326 engineers and not data scientists—**Threshy is not designed for hyper-parameter
4327 tuning of models**, but for threshold tuning of intelligent web services where internal
4328 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{}>.

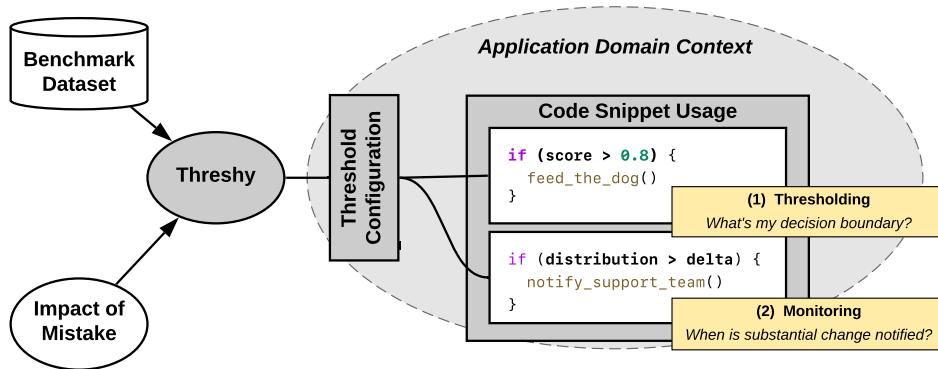


Figure 9.3: Threshy supports two key aspects for intelligent web services: threshold selection and monitoring.

4329 operates to fine-tune thresholds and explore trade-offs of prediction hits/misses. This
 4330 exposes the need for optimisation of thresholds, which is dependent on particular
 4331 use cases.

4332 Threshy improves developer productivity through automation of the threshold
 4333 selection process by leveraging an optimisation algorithm to propose thresholds.
 4334 Figure 9.3 illustrates the two key aspects by which Threshy can assist the developer's
 4335 application domain context. Developers input a representative dataset of their applica-
 4336 tion data (a benchmark dataset) in addition to cost factors to Threshy. Threshy's
 4337 output helps developers select appropriate thresholds within their applications and
 4338 can be used for monitoring if substantial change occurs within the service. The
 4339 algorithm considers different cost factors providing developers with summary infor-
 4340 mation so they can make more informed trade-offs. Developers also benefit from the
 4341 workflow implemented in Threshy by providing a reproducible procedure for testing
 4342 and tuning thresholds for any category of classification problem (binary, multi-class,
 4343 and multi-label). Threshy has also been designed to work for different input data
 4344 types including images, text and categorical values. The output, is a text file and
 4345 can be integrated into client applications ensuring that the thresholds can be up-
 4346 dated without code changes (if needed), and continuously monitored in a production
 4347 setting.

4348 9.2 Motivating Example

4349 As a motivating example consider Nina, a fictitious developer, who has been em-
 4350 ployed by Lucy's Tomato Farm to automate the picking of tomatoes from their vines
 4351 (when ripe) using computer vision and a harvesting robot. Lucy's Farm grow five
 4352 types of tomatoes (roma, cherry, plum, green, and yellow tomatoes). Nina's robot—
 4353 using an attached webcam—will crawl and take a photo of each vine to assess it
 4354 for harvesting. Nina's automated harvester needs to sort picked tomatoes into a
 4355 respective container, and thus several business rules need to be encoded into the
 4356 prediction logic to sort each tomato detected based on its *ripeness* (ripe or not ripe)

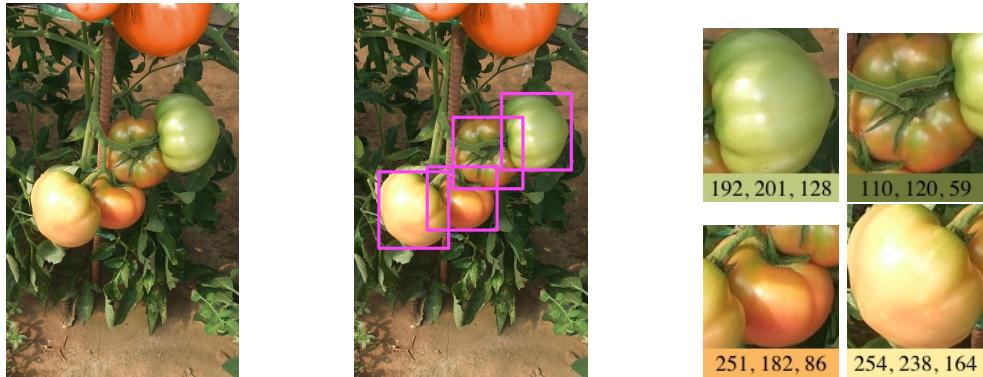


Figure 9.4: Pipeline of Nina’s harvesting robot. *Left:* Photo from harvesting robot’s webcam. *Centre:* Classification detecting different types of tomatoes. *Right:* Binary classification for ripeness (ripe/unripe) based on (R, G, B values).

4357 and *type of tomato* (as above).

4358 4359 4360 4361 Nina uses a two-stage pipeline consisting of a multi-class and a binary classification model. She has decided to evaluate the viability of cloud based intelligent services and use them if operationally effective. Figure 9.4 illustrates an example of the the pipeline as listed below:

- 4362 1. **Classify tomato ‘type’.** This stage uses an object localisation service to detect 4363 all tomato-like objects in the frame and classifies each tomato into one of the 4364 following labels: [‘roma’, ‘cherry’, ‘plum’, ‘green’, ‘yellow’].
- 4365 2. **Assess tomato ‘ripeness’.** This stage uses a crop of the localised tomatoes 4366 from the original frame to assess the crop’s colour properties (i.e., average 4367 colour must have $R > 200$ and $G < 240$). This produces a binary classification 4368 to deduce whether the tomato is ripe or not.

4369 4370 4371 4372 4373 4374 4375 4376 Nina only has a minimal appreciation of the evaluation method to use for off-the-shelf computer vision (classification) services. She also needs to consider the financial costs of mis-classifying either the tomato type or the ripeness. Missing a few ripe tomatoes isn’t a problem as the robot travels the field twice a week during harvest season. However, picking an unripe tomato is expensive as Lucy cannot sell them. Therefore, Nina needs a better (automated) way to assess the performance of the service and set optimal thresholds for her picking robot, thereby maximising profit.

4377 4378 4379 4380 4381 To assist in developing Nina’s pipeline, Lucy sampled a section of 1000 tomatoes by taking a photo of each tomato, labelling its type, and assessing whether the vine was ‘ripe’ or ‘not_ripe’. Nina ran the labelled images through an intelligent service, with each image having a predicted type (multi-class) and ripeness (binary), with respective confidence values.

4382 4383 4384 4385 Nina combined the predictions, their respective confidence values, and Lucy’s labelled ground truths into a CSV file which was then uploaded to Threshy. Nina asked Lucy to assist in setting relevant costs for correct predictions and false predictions. Threshy then recommended a choice of decision threshold which Nina then

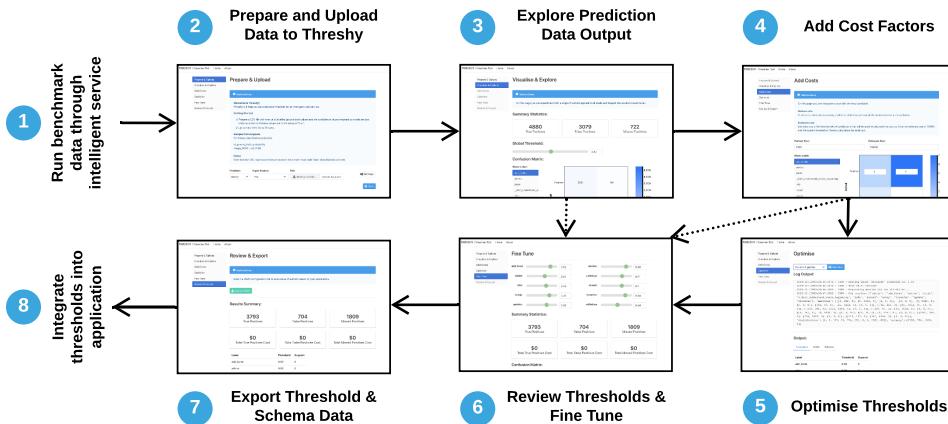


Figure 9.5: UI workflow for interacting with Threshy to optimise the thresholds for classification problem.

4386 fine tuned while considering the performance and cost implications.

4387 9.3 Threshy

4388 Threshy is a tool to assist software engineers with setting decision thresholds when
 4389 integrating machine-learnt components in a system. Our tool also serves as a method
 4390 to inform and educate engineers about the nuances to consider. The novel features
 4391 of Threshy are:

- 4392 • Automating threshold selection using an optimisation algorithm (NSGA-II
 4393 [90]), optimising the results for each label.
- 4394 • Support for additional user defined weights when optimising thresholds such
 4395 as financial costs and impact to society (different type of cost). This allows
 4396 decision thresholds to be set within a business context as they differ from
 4397 application to application [101].
- 4398 • Handles nuances of classification problems such as dealing with multi-objective
 4399 optimisation, and metric selection—reducing errors of omission.
- 4400 • Support key classification problems including binary (e.g. email is either
 4401 spam or ham), multi-class (e.g. predicting the colour of a car), and multi-label
 4402 (e.g. assign multiple topics to a document). Existing tools ignore multi-label
 4403 classification.

4404 Setting thresholds in Threshy is an eight step process as shown in Figure 9.5.
 4405 Software engineers ① run a benchmark dataset through the machine-learnt component
 4406 to create a CSV file with true labels and predicted labels along with the
 4407 predicted confidence values. The CSV file is then ② uploaded for initial exploration
 4408 where engineers can ③ experiment with modifying a single global threshold
 4409 for the dataset. Developers may choose to exit at this point (as indicated by dotted
 4410 arrows in Figure 9.5). Optionally, the engineer ④ defines costs for missed predictions
 4411 followed by selecting optimisation settings. The optional optimisation step of

4412 Threshy (5) considers the performance and costs when deriving the thresholds. Fi-
4413 nally, the engineer can (6) review and fine tune the calculated thresholds, associated
4414 costs, and (7) download generated threshold meta-data to be (8) integrated into their
4415 application.

4416 Threshy runs a client/server architecture with a thin-client (see Figure 9.6). The
4417 web-based application consists of an interactive front-end where developers upload
4418 benchmark results—consisting of both human annotated labels (ground truths) and
4419 machine predictions (from the intelligent service)—and use threshold tuners (via
4420 sliders) to present a data summary of the uploaded CSV. Predicted performances
4421 and costs are entered manually into the web interface by the developer. The back-end
4422 of Threshy asynchronously runs a data analyser, cost processor and metrics calculator
4423 when relevant changes are made to the front-end’s tuning sliders. Separating the
4424 two concerns allows for high intensity processing to be done on the server and not
4425 the front end.

4426 The data analyser provides a comprehensive overview of confusion matrices
4427 compatible for multi-label multi-class classification problems. When representing
4428 the confusion matrix, it is trivial to represent instances where multi-label multi-
4429 classification is not considered. For example, in the simplest case, a single row in
4430 the matrix represents a single label out of two classes, or each row has one label but
4431 it has multiple classes. However, a more challenging case to visualise the confusion
4432 arises when you have n labels and n classes; the true/false matches become too
4433 excessive to visualise as it is disproportionate to the true results. To deal with this
4434 issue, we condense the summary statistics down to three constructs: (i) number of
4435 true positives, (ii) false positives, (iii) missed positives. This therefore allows us to
4436 optimise against the true positives and minimise the other two constructs.

4437 Threshy is a fully self-contained repository containing implementation of the
4438 tool, scripting and exploratory notebooks, which we make available at <https://github.com/a2i2/threshy>.

4440 9.4 Related work

4441 9.4.1 Decision Boundary Estimation

4442 Optimal machine-learnt decision boundaries depend on identifying the operating
4443 conditions of the problem domain. A systematic study by Drummond and Holte
4444 [101] classifies four such operating conditions to determine a decision threshold: (i)
4445 the operating condition is known and thus the model trained matches perfectly; (ii)
4446 where the operating conditions are known but change with time, and thus the model
4447 must be adaptable to such changes; (iii) where there is uncertainty in the knowledge
4448 of the operating conditions certain changes in the operating condition are more likely
4449 than others; (iv) where there is no knowledge of the operating conditions and the
4450 conditions may change from the model in any possible way. Various approaches
4451 to determine appropriate thresholds exist for all four of these cases, such as cost-
4452 sensitive learning, ROC analysis, cost curves, and Brier scores.

4453 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 [294], 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.

9.4.2 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 [147, 254].) 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 9.1) to reflect the nature of real-world development. Google’s *What-If Tool* [345] 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 [146] provides an interface to test hypotheticals (although only on Generalized Additive Models) and their *ModelTracker* tool [9] 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.

9.5 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.

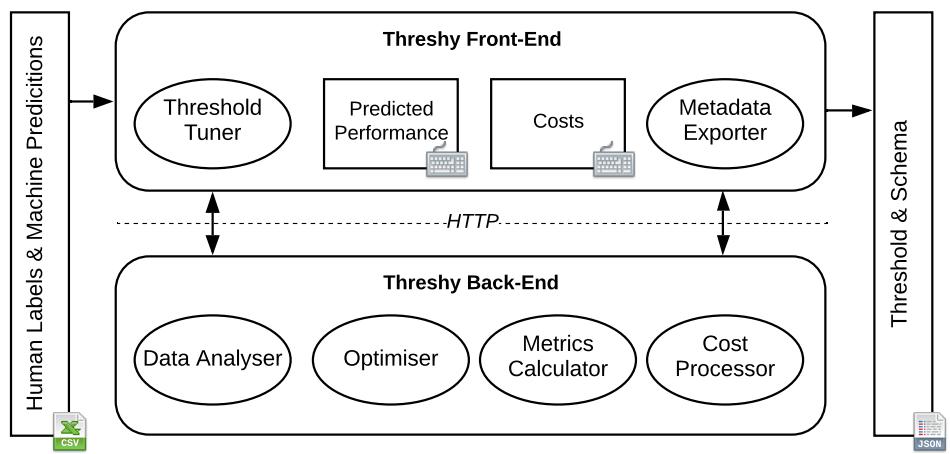


Figure 9.6: Architecture of Threshy.

CHAPTER 10

4491

4492

4493 An Integration Architecture Tactic to guard AI-first Components[†]

4494

4495 **Abstract** Intelligent web services provide the power of AI to developers via simple REST-
4496 ful API endpoints, abstracting away many complexities of machine learning. However,
4497 most of these intelligent web services (IWSs)—such as computer vision—continually learn
4498 with time. When the internals within the abstracted ‘black box’ become hidden and evolve,
4499 pitfalls emerge in the robustness of applications that depend on these evolving services.
4500 Without adapting the way developers plan and construct projects reliant on IWSs, signifi-
4501 cant gaps and risks result in both project planning and development. Therefore, how can
4502 software engineers best mitigate software evolution risk moving forward, thereby ensuring
4503 that their own applications maintain quality? Our proposal is an architectural tactic designed
4504 to improve intelligent service-dependent software robustness. The tactic involves creating
4505 an application-specific benchmark dataset baselined against an intelligent service, enabling
4506 evolutionary behaviour changes to be mitigated. A technical evaluation of our implemen-
4507 tation of this architecture demonstrates how the tactic can identify 1,054 cases of substantial
4508 confidence evolution and 2,461 cases of substantial changes to response label sets using a
4509 dataset consisting of 331 images that evolve when sent to a service.

4510 10.1 Introduction

4511 The introduction of intelligent web services (IWSs) into the software engineering
4512 ecosystem allows developers to leverage the power of artificial intelligence (AI)
4513 without implementing complex AI algorithms, source and label training data, or
4514 orchestrate powerful and large-scale hardware infrastructure. This is extremely
4515 enticing for developers to embrace due to the effort, cost and non-trivial expertise
4516 required to implement AI in practice [265, 295].

[†]This chapter is originally based on A. Cummaudo, S. Barnett, R. Vasa, J. Grundy, and M. Abdellrazek, “Beware the evolving ‘intelligent’ web service! An integration architecture tactic to guard AI-first components,” 2020, Unpublished. Terminology has been updated to fit this thesis.



'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 10.1: 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.

4517 However, the vendors that offer these services also periodically update their
4518 behaviour (responses). The ideal practice for communicating the evolution of a
4519 web service involves updating the version number and writing release notes. The
4520 release notes typically describe new capabilities, known problems, and requirements
4521 for proper operation [45]. Developers anticipate changes in behaviour between ver-
4522 sioned releases although they expect the behaviour of a specific version to remain
4523 stable over time [332]. However, emerging evidence indicates that ‘intelligent’ ser-
4524 vices *do not* communicate changes explicitly [80]. Intelligent services evolve in
4525 unpredictable ways, provide no notification to developers and changes are undocu-
4526 mented [84]. To illustrate this, consider Figure 10.1, which shows the evolution of a
4527 popular computer vision service (CVS) with examples of labels and associated confi-
4528 dence scores changing are shown. This behaviour change severely negatively affects
4529 reliability. Applications may no longer function correctly if labels are removed or
4530 confidence scores change beyond predefined thresholds.

4531 Unlike traditional web services, the functionality of these IWSs is dependent
4532 on a set of assumptions unique to their machine learning principles and algorithms.
4533 These assumptions are based on the data used to train machine learning algorithms,
4534 the choice of algorithm, and the choice of data processing steps—most of which
4535 are not documented to service end users. The behaviour of these services evolve
4536 over time [81]—typically this implies the underlying model has been updated or
4537 re-trained.

4538 Vendors do not provide any guidance on how best to deal with this evolution in
4539 client applications. For developers to discover the impact on their applications they
4540 need to know the behavioural deviation and the associated impact on the robustness
4541 and reliability of their system. Currently, there is no guidance on how to deal with
4542 this evolution, nor do developers have an explicit checklist of the likely errors and
4543 changes that they must test for [84].

4544 In this paper, we present a reference architecture to detect the evolution of such
4545 IWSs. This tactic can be used both by intelligent service consumers, to defend their
4546 applications against the evolutionary issues present in IWSs, and by service vendors
4547 to make their services more robust. We also present a set of error conditions that
4548 occur in existing CVSs.

4549 The key contributions of this paper are:

- 4550 • A set of new service error codes for describing the empirically observed error
4551 conditions in IWSs.
- 4552 • A new reference architecture for using IWSs with a Proxy Server that returns
4553 error codes based on an application specific benchmark dataset.
- 4554 • A labelled data set of evolutionary patterns in CVSs.
- 4555 • An evaluation of the new architecture and tactic showing its efficacy for
4556 supporting IWS evolution from both provider and consumer perspectives.

4557 The rest of this paper is organised thus: Section 10.2 presents a motivating
4558 example that anchors our work; Section 10.3 presents a landscape analysis on IWSs;
4559 Section 10.4 presents an overview of our architecture; Section 10.5 describes the
4560 technical evaluation; Section 10.6 presents a discussion into the implications of our

4561 architecture, its limitations and potential future work; Section 10.7 discusses related
4562 work; Section 10.8 provides concluding remarks.

4563 10.2 Motivating Example

4564 We identify the key requirements for managing evolution of IWSs using a motivating
4565 example. Consider Michelina, a software engineer tasked with developing a fall
4566 detector system for helping aged care facilities respond to falls promptly. Michelina
4567 decides to build the fall detector with an intelligent service for detecting people as she
4568 has no prior experience with machine learning. The initial system built by Michelina
4569 consists of a person detector and custom logic to identify a fall based on rapid shape
4570 deformation (i.e., a vertical ‘person’ changing to a horizontal ‘person’ greater than
4571 specified probability threshold value). Due to the inherent uncertainty present in
4572 an intelligent service and the importance of correctly identifying falls, Michelina
4573 informs the aged care facility that they should manually verify falls before dispatching
4574 a nurse to the location. The aged care facility is happy with this approach but inform
4575 Michelina that only a certain percentage of falls can be manually verified based on
4576 the availability of staff. In order to reduce the manual work Michelina sets thresholds
4577 for a range of confidence scores where the system is uncertain. Michelina completes
4578 the fall detector using a well-known cloud-based intelligent image classification web
4579 service and her client deploys this new fall detection application.

4580 Three months go by and then the aged care facility contact Michelina saying the
4581 percentage of manual inspections is far too high and could she fix it. Michelina is
4582 mystified why this is occurring as she thoroughly tested the application with a large
4583 dataset provided by the aged care facility. On further inspection Michelina notices
4584 that the problem is caused by some images classifying the person with a ‘child’
4585 label rather than a ‘person’ label. Michelina is frustrated and annoyed at this
4586 behaviour as (i) the cloud vendor did not document or notify her of the change of the
4587 intelligent service behaviour, (ii) she does not know the best practice for dealing with
4588 such a service evolution, and (iii) she cannot predict how the service will change
4589 in the future. This experience also makes Michelina wonder what other types of
4590 evolution can occur and how can she minimise these behavioural changes on her
4591 critical care application. Michelina then begins building an ad-hoc solution hoping
4592 that what she designs will be sufficient.

4593 For Michelina to build a robust solution she needs to support the following
4594 requirements:

- 4595 **R1.** Define a set of error conditions that specify the types of evolution that occur
4596 for an intelligent service.
- 4597 **R2.** Provide a notification mechanism for informing client applications of be-
4598 havioural changes to ensure the robustness and reliability of the application.
- 4599 **R3.** Monitor the evolution of IWSs for changes that affect the application’s be-
4600 haviour.
- 4601 **R4.** Implement a flexible architecture that is adaptable to different IWSs and ap-
4602 plication contexts to facilitate reuse.

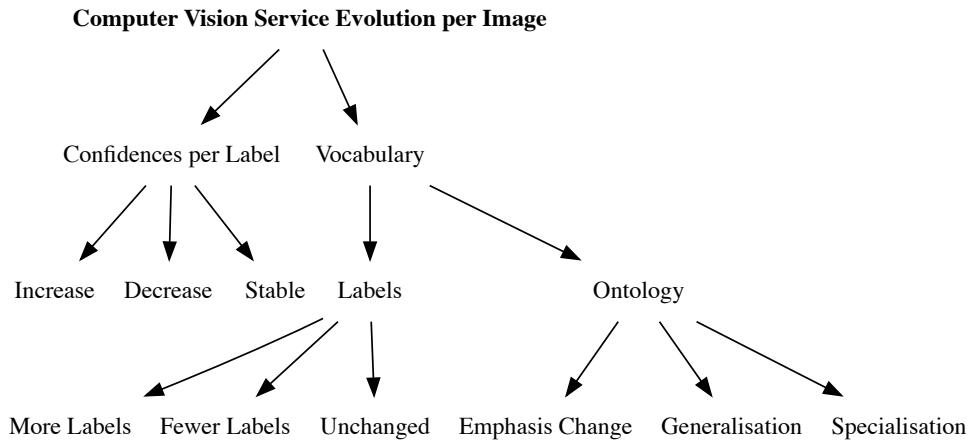


Figure 10.2: The dimensions of evolution identified within CVSs.

4603 **10.3 Intelligent Services**

4604 We present background information on IWSs describing how they differ from tra-
 4605 ditional web services, the dimensions of their evolution and the currently limited
 4606 configuration options available to users.

4607 **10.3.1 ‘Intelligent’ vs ‘Traditional’ Web Services**

4608 Unlike conventional web services, IWSs are built using AI-based components. These
 4609 components are unlike traditional software engineering paradigms as they are data-
 4610 dependent and do not result in deterministic outcomes. These services make future
 4611 predictions on new data based solely against its training dataset; outcomes are
 4612 expressed as probabilities that the inference made matches a label(s) within its
 4613 training data. Further, these services are often marketed as forever evolving and
 4614 ‘improving’. This means that their large training datasets may continuously update
 4615 the prediction classifiers making the inferences, resulting both in probabilistic and
 4616 non-deterministic outcomes [81, 149]. Critically for software engineers using the
 4617 services, these non-deterministic aspects have not been sufficiently documented in
 4618 the service’s API documented, which has been shown to confuse developers [84].

4619 A strategy to combat such service changes, which we often observe in traditional
 4620 software engineering practices, are for such services to be versioned upon substantial
 4621 change. Unfortunately emerging evidence indicates that prominent cloud vendors
 4622 providing these IWSs do not release new versioned endpoints of the APIs when the
 4623 *internal model* changes [81]. For IWSs, it is impossible to invoke requests specific
 4624 to a particular version model that was trained at a particular date in time. This means
 4625 that developers need to consider how evolutionary changes to the IWSs they make
 4626 use of may impact their solutions *in production*.



Figure 10.3: A significant confidence increase ($\delta = +0.425$) from ‘window’ (0.559) to ‘water transportation’ (0.984) goes beyond simple decision boundaries.

4627 10.3.2 Dimensions of Evolution

4628 The various key dimensions of the evolution of IWSs is illustrated in Figure 10.2.
 4629 There are two primary dimensions of evolution: *changes to the label sets* returned
 4630 per image submitted and *changes to the confidences* per label in the set of labels
 4631 returned per image. In the former, we identify two key aspects: cardinality changes
 4632 and ontology changes. Cardinality changes occur when the service either introduces
 4633 or drops a label for the same image at two different generations. Alternatively, the
 4634 cardinality may remain stagnant, although this is not guaranteed. This results in
 4635 an expectation mismatch by developers as to what labels can or will be returned by
 4636 the service. For instance, the terms ‘black’ and ‘black and white’ may be found to
 4637 be categorised as two separate labels. Secondly, the ontologies of these labels are
 4638 non-static, and a label may become more generalised into a hypernym, specialised
 4639 into a hyponym, or the emphasis of the label may change either to a co-hyponym or
 4640 another aspect in the image, such as the colour or scene, rather than the subject of
 4641 the image [81].

4642 Secondly, we have identified that the confidence values returned per label are also
 4643 non-static. While some services may present minor changes to labels’ confidences
 4644 resulting from statistical noise, other labels had significant changes that were beyond
 4645 basic decision boundaries. An example is shown in Figure 10.3. Developer code
 4646 written to assume certain ranges/confidence intervals will fail if the service evolves
 4647 in this way.

4648 10.3.3 Limited Configurability

4649 As an example, consider Figure 10.5, which illustrates an image of a dog uploaded
 4650 to a well-known cloud-based CVS. Developers have very few configuration param-
 4651 eters in the upload payload (`url` for the image to analyse and `maxResults` for the
 4652 number of objects to detect). The JSON output payload provides the confidence
 4653 value of its estimated bounding box and label of the dog object via its `score` field
 4654 (0.792). Developers can only modify these parameters to influence the score to
 4655 improve the performance of the IWS. This is unlike many machine learning toolkit
 4656 hyper-parameter optimisation facilities, which can be used to configure the internal
 4657 parameters of the algorithm for training a model. In this case, developers using the
 4658 IWS have no insight into which hyperparameters were used when training the model

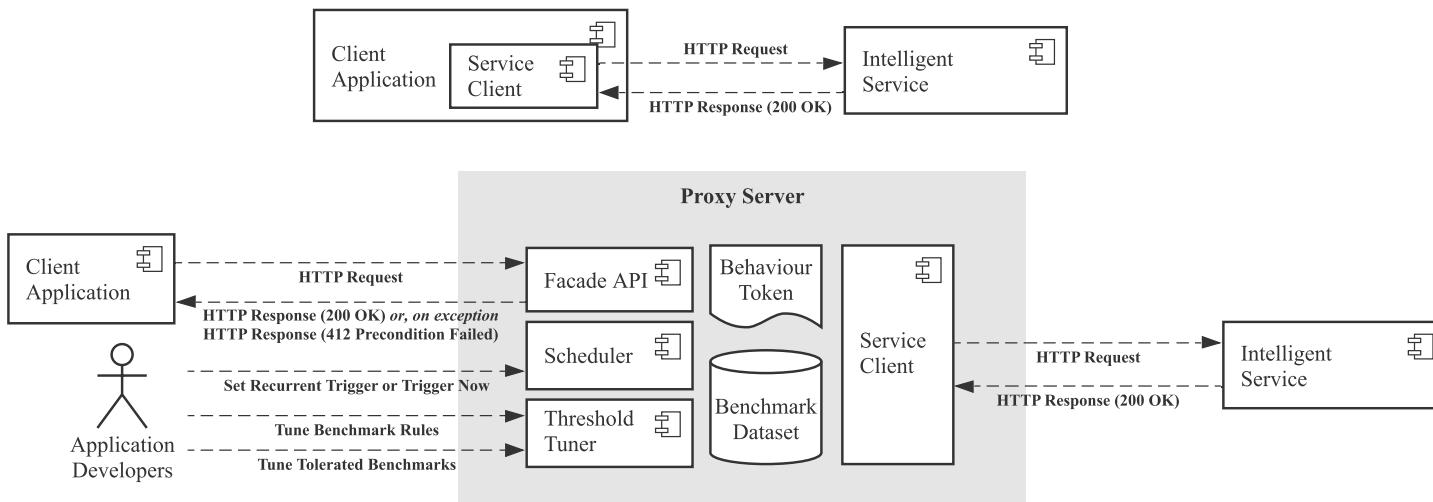


Figure 10.4: Top: Accessing an intelligent service directly. Bottom: Primary components of the Proxy Server approach.

4659 or the algorithm selected, and cannot tune the trained model. Thus an evaluation
 4660 procedure must be followed as a part of using an intelligent service for an application
 4661 to tune their output confidence values.

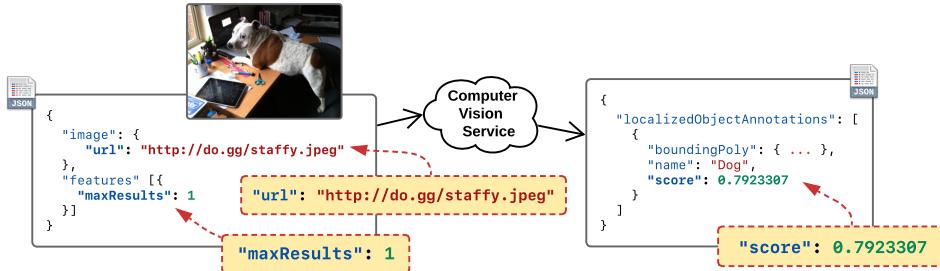


Figure 10.5: Request and response for an intelligent computer vision web service with only three configuration parameters: the image's url, maxResults and score.

4662 However, decision boundaries in service client code using simple If conditions
 4663 around confidence scores is not a sufficient enough strategy, as evidence shows intel-
 4664 ligent, non-deterministic web services change sporadically and unknowingly. Most
 4665 traditional, deterministic code bases handle unexpected behaviour of called APIs via
 4666 *error codes* and exception handling. Thus the non-deterministic components of the
 4667 client code, such as those using CVSs, will also tend to conflict with their traditional
 4668 deterministic components as the latter do not deal in terms of probabilities but in
 4669 using error codes. This makes achieving robust component integration in client code
 4670 bases hard. More sophisticated monitoring of IWSs in client code is therefore re-
 4671 quired to map the non-deterministic service behaviour changes to errors such that the
 4672 surrounding infrastructure can support it and reduce interface boundary problems.
 4673 While data science literature acknowledges the need for such an architecture [105]
 4674 they do not offer any technical software engineering solutions to mitigate the issues
 4675 such that software engineers have a pattern to work against it. To date, there do not
 4676 yet exist IWS client code architectures, tactics or patterns that achieve this goal.

4677 10.4 Our Approach

4678 To address the requirements from Section 10.2 we have developed a new Proxy
 4679 Service¹ that includes: (i) evaluation of an intelligent service using an application
 4680 specific benchmark dataset, (ii) a Proxy Server to provide client applications with
 4681 evolution aware errors, and (iii) a scheduled evolution detection mechanism. The
 4682 current approach of using an intelligent API via direct access is shown in Figure 10.4
 4683 (top). In contrast, an overview of our approach is shown in Figure 10.4 (bottom).
 4684 The following sections describe our approach in detail.

¹A reference architecture is provided at <http://bit.ly/2TlMmDh>.

Table 10.1: 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.

4685 10.4.1 Core Components

4686 For the purposes of this paper we assume that the intelligent service of interest
4687 is an image recognition service, but our approach generalises to other intelligent,
4688 trained model-based services e.g., natural language processing, document recogni-
4689 tion, voice, etc. Each image, when uploaded to the intelligent service returns a
4690 response (R) which is a set describing a label (l) of what is in the image (i) along
4691 with its associated confidence (c)—thus $R_i = \{(l_1, c_1), (l_2, c_2), \dots, (l_n, c_n)\}$. Most
4692 documentation of these services imply that these confidence values are all what is
4693 needed to handle evolution in their systems. This means that if a label changes
4694 beyond a certain threshold, then the developer can deal with the issue then (or ignore
4695 it). While this approach may work in some simple application contexts, in many it
4696 may not. Our Proxy Server offers a way to monitor if these changes go beyond a
4697 threshold of tolerance, checking against a domain-specific dataset over time.

4698 10.4.1.1 Benchmark Dataset

4699 Monitoring an intelligent service for behaviour change requires a Benchmark Dataset,
4700 a set of n images. For each image (i) in the Benchmark Dataset (B) there is an associ-
4701 ated label (l) that represents the true value for that item; $B_i = \{(i_1, l_1), (i_2, l_2), \dots, (i_n, l_n)\}$.

Table 10.2: 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.

⁴⁷⁰² This dataset is used to check for evolution in IWSs. By using a dataset specific to the
⁴⁷⁰³ application domain, developers can detect when evolution affects their application
⁴⁷⁰⁴ rather than triggering all non-impactful changes. This helps achieve our require-
⁴⁷⁰⁵ ment *R3. Monitor the evolution of IWSs for changes that affect the application's*
⁴⁷⁰⁶ *behaviour.* Using application-specific datasets also ensures that the architectural
⁴⁷⁰⁷ style can be used for different IWSs as only the data used needs to change. This
⁴⁷⁰⁸ design choice encourages reuse satisfying requirement *R4. Implement a flexible*
⁴⁷⁰⁹ *architecture that is adaptable to different IWSs and application contexts to facilitate*
⁴⁷¹⁰ *reuse.*

⁴⁷¹¹ 10.4.1.2 Facade API

⁴⁷¹² An architectural ‘facade’ is the central component to our mitigation strategy for
⁴⁷¹³ monitoring and detecting for changes in called IWSs. The facade acts as a guarded
⁴⁷¹⁴ gateway to the intelligent service that defends against two key issues: (i) potential
⁴⁷¹⁵ shifts in model variations that power the cloud vendor services, and (ii) ensures that
⁴⁷¹⁶ a context-specific dataset specific to the application being developed is validated
⁴⁷¹⁷ over time. By using a facade we can return evolution-aware error codes to the client
⁴⁷¹⁸ application satisfying requirement *R1. Define a set of error conditions that specify*
⁴⁷¹⁹ *the types of evolution that occur for an intelligent service* and enabling requirement
⁴⁷²⁰ *R3. Monitor the evolution of IWSs for changes that affect the application's behaviour.*

⁴⁷²¹ 10.4.1.3 Threshold Tuner

⁴⁷²² Selecting an appropriate threshold for detecting behavioural change depends on the
⁴⁷²³ application context. Setting the threshold too low increases the likelihood of incor-
⁴⁷²⁴ rect results, while setting the threshold too high means undesired changes are being
⁴⁷²⁵ detected. Our approach enables developers to configure these parameters through a
⁴⁷²⁶ Threshold Tuner. This improves robustness as now there is a systematic approach for
⁴⁷²⁷ monitoring and responding to incorrect thresholds. Configurable thresholds meet
⁴⁷²⁸ our key requirements *R2* and *R3*.

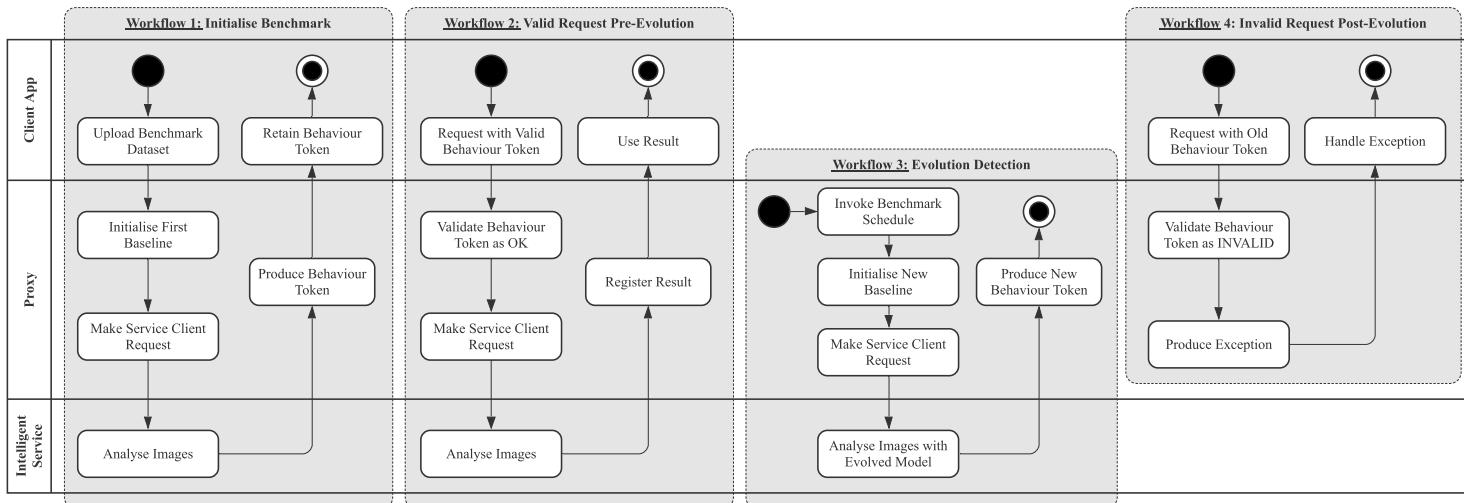


Figure 10.6: State diagram for the four workflows presented.

4729 *10.4.1.4 Behaviour Token*

4730 The Behaviour Token stores the current state of the Proxy Server by encoding specific
4731 rules regarding the evolution of the intelligent service. The current token (at time t)
4732 held by the Proxy Server is denoted by k_t . These rules are specified by the developer
4733 upon initialisation of this Proxy Server, and are presented in Table 10.2. When the
4734 Proxy Server is first initialised (i.e., at $t = 0$), the first Behaviour Token is created
4735 based on the Benchmark Dataset and its configuration parameters (Table 10.2) and
4736 is stored locally (thus k_0 is created). The Behaviour Token is passed to the client
4737 application to be used in subsequent requests to the proxy server, where k_r represents
4738 the Behaviour Token passed from the client application to the proxy server. Each
4739 time the proxy server receives the Behaviour Token from the client the validity of the
4740 token is validated with a comparison to the Proxy Server's current behaviour token
4741 (i.e., $k_r \equiv k_t$). An invalid token (i.e., when $k_r \neq k_t$) indicates that an error caused by
4742 evolution has occurred and the application developer needs to appropriately handle
4743 the exception. Behaviour Tokens are essential for meeting requirement *R3. Monitor*
4744 *the evolution of IWSs for changes that affect the application's behaviour.*

4745 *10.4.1.5 Service Client*

4746 If any of the rules above are violated, then the response of the facade request will
4747 vary depending on the parameter of the behaviour encoded within the behaviour
4748 token. This can be one of:

- 4749** • **Error:** Where a HTTP non-200 code is returned by the facade to the client
4750 application, indicating that the client application must deal with the issue
4751 immediately;
- 4752** • **Warning:** Where a warning ‘callback’ endpoint is called with the violated
4753 response to be dealt with, but the response is still returned to the client
4754 application;
- 4755** • **Info:** Where the violated response is logged in the facade’s logger for the
4756 developer to periodically read and inspect, and the response is returned to the
4757 client application.

4758 We implement this Proxy Server pattern using HTTP conditional requests. As
4759 we treat the Label as a first class citizen, we return the labels for a specific image
4760 (r_i) only where the *Entity Tag* (ETag) or *Last Modified* validators pass. The k_r
4761 is encoded within either the ETag (i.e., a unique identifier representing t) or as
4762 the date labels (and thus models) were last modified (i.e., using the *If-Match*
4763 or *If-Unmodified-Since* conditional headers). We note that the use of *weak*
4764 ETags should be used, as byte-for-byte equivalence is not checked but only semantic
4765 equivalence within the tolerances specified. Should t evolve to an invalid state
4766 (i.e., k_r is no longer valid against k_t) then the behaviour as described above will be
4767 enacted.

4768 These HTTP header fields are used as the ‘backbone’ to help enforce robustness
4769 of the services against evolutionary changes and context within the problem domain
4770 dataset. Responses from the service are forwarded to the clients when such rules

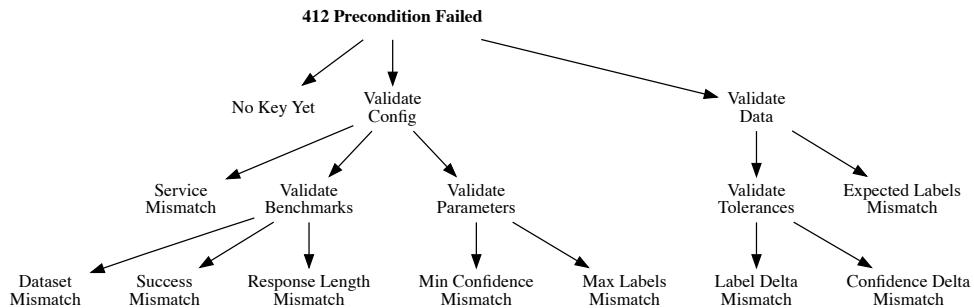


Figure 10.7: Precondition failure taxonomy; leaf nodes indicate error types returned to users.

are met, otherwise alternative behaviour occurs. For example, the most severe of violated erroneous behaviour is the ‘Error’ behaviour. To enforce this rule, we advocate for use of the 412 Precondition Failed HTTP error if a violation occurs, as a `If-*` conditional header was violated. An example of this architectural pattern with the ‘Error’ behaviour is illustrated in Figure 10.6.

We suggest the 412 Precondition Failed HTTP error be returned in the event that a behaviour token is violated against a new benchmark. Further details outlining the reasons why a precondition has failed are encoded within a JSON response sent back to the consuming application. The following describes the two broad categories of possible errors returned: *robustness precondition failure* or *benchmark precondition failure*. These are illustrated in a high level within Figure 10.7 where leaf nodes are the potential error types that can be returned. A list of the different error codes are given in Table 10.1, where errors above the rule are robustness expectations (which check for basic requirements such as whether the key provided encodes the same data as the dataset in the facade) while those below are benchmark expectations (which identifies evolution cases).

10.4.1.6 Scheduler

The Scheduler is responsible for triggering the Evolution Detection Workflow (described in detail below in Section 10.4.2). Developers set the schedule to run in the background at regular intervals or to trigger if violations occur z times. The Scheduler is the component that enables our architectural style to identify called intelligent service software evolution and to notify the client applications that such evolution has occurred. Client applications can then respond to this evolution in a timely manner rather than wait for the system to fail, as in our motivating example. The Scheduler is necessary to satisfy our requirements *R2* and *R3*.

10.4.2 Usage Example

We explain how developer Michelina, from our motivating example, would use our proposed solution to satisfy the requirements described in Section 10.2. Each workflow is presented in Figure 10.6. Only *Workflow 1 - Initialise Benchmark* is

4800 executed once, while the rest are cycled. The description below assumes Michelina
4801 has implemented the Proxy.

4802 10.4.2.1 *Workflow 1. Initialise Benchmark*

4803 The first task that Michelina has to do is to prepare and initialise the benchmark
4804 dataset within the Proxy Server. To prepare a representative dataset, Michelina needs
4805 to follow well established guidelines such as those proposed by Pyle. Michelina also
4806 needs to manually assign labels to each image before uploading the dataset to the
4807 Proxy along with the thresholds to use for detecting behavioural change. The full set
4808 of parameters that Michelina has to set are based on the rules shown in Table 10.2.
4809 Michelina cannot use the Proxy to notify her of evolution until a Benchmark Dataset
4810 has been provided. The Proxy then sends each image in the Benchmark Dataset to
4811 the intelligent service and stores the results. From these results, a Behaviour Token
4812 is generated which is passed back to the Client Application. Michelina uses this
4813 token in all future requests to the Proxy as the token captures the current state of the
4814 intelligent service.

4815 10.4.2.2 *Workflow 2. Valid Request Pre-Evolution*

4816 Workflow 2 represents the steps followed when the intelligent service is behaving as
4817 expected. Michelina makes a request to label an image to the Proxy using the token
4818 that she received when registering the Benchmark Dataset. The token is validated
4819 with the Proxy's current state token and then a request to label the image is made to
4820 the intelligent service if no errors have occurred. Results returned by the intelligent
4821 service are registered with the Proxy Server. Michelina can be confident that the
4822 result returned by our service is in line with her expectations.

4823 10.4.2.3 *Workflow 3. Evolution Detection*

4824 Workflow 3 describes how the Proxy functions when behavioural change is present
4825 in the called intelligent service. Michelina sets a schedule for once a day so that the
4826 Proxy's Scheduler triggers Workflow 3. First, each image in the Benchmark Dataset
4827 is sent to the intelligent service. Unlike, Workflow 1, we already have a Behaviour
4828 Token that represents the previous state of the intelligent service. In this case, the
4829 model behind the intelligent service has been updated and provides different results
4830 for the Benchmark Dataset. Second, the Proxy updates the internal Behaviour Token
4831 ready for the next request. At this stage Michelina will be notified that the behaviour
4832 of the intelligent service has changed.

4833 10.4.2.4 *Workflow 4. Invalid Request Post-Evolution*

4834 Workflow 4 provides Michelina with an error message when evolution has been
4835 detected. Michelina's client application makes a request to the Proxy Server with
4836 an old Behaviour Token. The Proxy Server then validates the client token which is
4837 invalid as the Behaviour Token has been updated. In this case, an exception is raised
4838 and an appropriate error message as discussed above is included in the response

4839 back to Michelina’s client application. Michelina can code her application to handle
4840 each error class in appropriate ways for her domain.

4841 10.5 Evaluation

4842 Our evaluation of our novel intelligent service Proxy Server approach uses a technical
4843 evaluation based on the results of an observational study. We used existing datasets
4844 from observational studies [81, 200] to identify problematic evolution in computer
4845 vision labelling services. Based on our findings we proposed and implemented the
4846 Proxy Server using a Ruby development framework which we have made available
4847 online for experimentation.² Additional data was collected from the CVS and sent
4848 to the Proxy Server to evaluate how the service handles behavioural change.

4849 10.5.1 Data Collection and Preparation

4850 To minimise reviewer bias, we do not identify the name of the service used, however
4851 this service was one of the most adopted cloud vendors used in enterprise applications
4852 in 2018 [277]. The two existing datasets used [81, 200] consisted of 6,680 images.

4853 We initialised the benchmark (workflow 1) in November 2018, and sent each
4854 image to the service every eight days and captured the JSON responses through the
4855 facade API (workflow 2) until March 2019. This resulted in 146,960 JSON responses
4856 from the target CVS. We then selected the first and last set of JSON responses (i.e.,
4857 13,360 responses) and independently identified 331 cases of evolution of the original
4858 6,680 images. This was achieved by analysing the JSON responses for each image
4859 taken in using an evaluation script.³

4860 For each JSON response, evolution (as classified by Figure 10.2) was determined
4861 either by a vocabulary or confidence per label change in the first and last responses
4862 sent. For the 331 evolving responses, we calculated the delta of the label’s confidence
4863 between the two timestamps and the delta in the number of labels recorded in the
4864 entire response. Further, for the highest-ranking label (by confidence), we manually
4865 classified whether its ontology became more specific, more generalised or whether
4866 there was substantial emphasis change. The distribution of confidence differences per
4867 these three groups are shown in Figure 10.8, with the mean confidence delta indicated
4868 with a vertical dotted line. This highlights that, on average, labels that change
4869 emphasis generally have a greater variation, such as the example in Figure 10.3.
4870 Further, we grouped each image into one of four broad categories—*food*, *animals*,
4871 *vehicles*, *humans*—and assessed the breakdown of ontology variance as provided
4872 in Table 10.3. We provide this dataset as an additional contribution and to permit
4873 replication.⁴ The parameters set for our initial benchmark were a delta label value of
4874 3 and delta confidence value of 0.01. Expected labels for relevant groups were also
4875 assigned as mandatory label sets (e.g., *animal* images used ‘animal’, ‘fauna’ and
4876 ‘organism’; *human* images used ‘human’ etc.).

²<http://bit.ly/2TIMmDh> last accessed 5 March 2020.

³<http://bit.ly/2G7saFJ> last accessed 2 March 2020.

⁴<http://bit.ly/2VQrAUU> last accessed 5 March 2020.

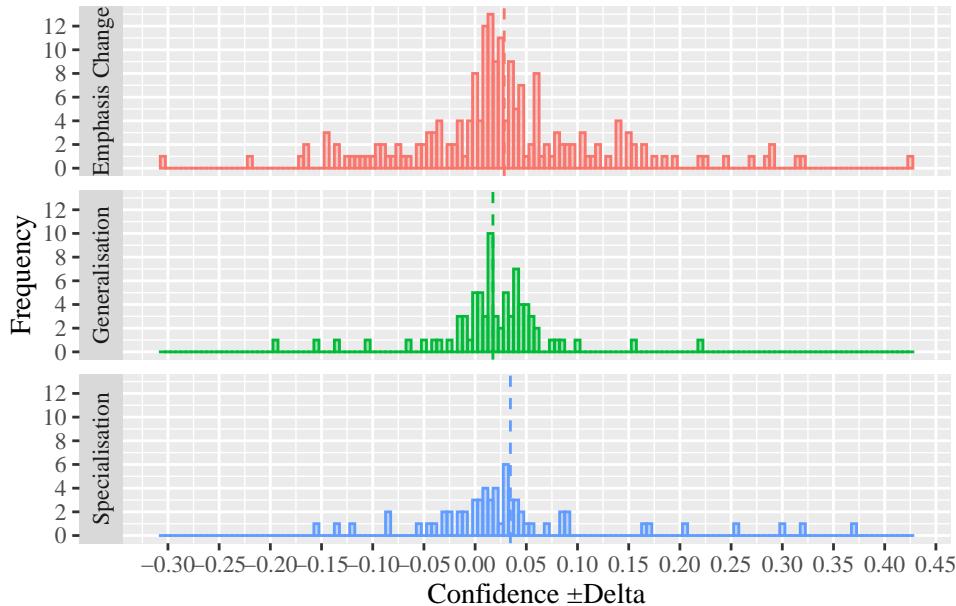


Figure 10.8: Histogram of confidence variation.

10.5.2 Results

Examples of the March 2019 responses contrasting the proxy and direct service responses in our evaluation are shown in Figures 10.9 to 10.11. (Due to space limitations, the entire JSON response is partially redacted using ellipses.) These examples identify the label identified with the highest level of confidence in three examples against the ground truth label in the benchmark dataset. In total, the Proxy Server identified 1,334 labels added to the responses and 1,127 labels dropped, with, on average, a delta of 8 labels added. The topmost labels added were ‘architecture’ at 32 cases, ‘building’ at 20 cases and ‘ingredient’ at 20 cases; the topmost labels dropped were ‘tree’ at 21 cases, ‘sky’ at 19 cases and ‘fun’ at 17 cases. 1054 confidence changes were also observed by the Proxy Server, on average a delta increase of 0.0977.

In Figure 10.9, we highlight an image of a sheep that was identified as a ‘sheep’ (at 0.9622) in November 2018 and then a ‘mammal’ in March 2019. This evolution was classified by the Proxy Server as a confidence change error as the delta in the confidences between the two timestamps exceeds the parameter set of 0.01—in this case, ‘sheep’ was downgraded to the third-ranked label at 0.9816, thereby increasing by a value of 0.0194. As shown in the example, four other labels evolved for this image between the two time stamps (‘herd’, ‘livestock’, ‘terrestrial animal’ and ‘snout’) with an average increase of 0.1174 found. Such information is encoded as a 412 HTTP error returned back to the user by the Proxy Server, rejecting the request as substantial evolution has occurred, however the response directly from the service indicates no error at all (indicating by a 200 HTTP response).

Similarly, Figure 10.10 shows a violation of the number of acceptable changes in

Table 10.3: 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

4901 the number of labels a response should have between two timestamps. In November
 4902 2018, the response includes the labels ‘car’, ‘motor vehicle’, ‘city’ and
 4903 ‘road’, however these labels are not present in the 2019 response. The response
 4904 in 2019 introduces ‘transport’, ‘building’, ‘architecture’, and ‘house’.
 4905 Therefore, the combined delta is 4 dropped and 4 introduced labels, exceeding our
 4906 threshold set of 3.

4907 Lastly, Figure 10.11 indicates an expected label failure. In this example, the
 4908 label ‘fauna’ was dropped in the 2018 label set, which was an expected label
 4909 of all animals we labelled in our dataset. Additionally, this particular response
 4910 introduced ‘green iguana’, ‘iguanidae’, and ‘marine iguana’ to its label
 4911 set. Therefore, not only was this response in violation of the label delta mismatch, it
 4912 was also in violation of the expected labels mismatch error, and thus is caught twice
 4913 by the Proxy Server.

4914 10.5.3 Threats to Validity

4915 10.5.3.1 Internal Validity

4916 As mentioned, we selected a popular CVS provider to test our proxy server against.
 4917 However, there exist many other CVSs, and due to language barriers of the authors,
 4918 no non-English speaking service were selected despite a large number available from
 4919 Asia. Further, no user evaluation has been performed on the architectural tactic so
 4920 far, and therefore developers may suggest improvements to the approach we have
 4921 taken in designing our tactic. We intend to follow this up with a future study.

4922 10.5.3.2 External Validity

4923 This paper only evaluates the object detection endpoint of a computer vision-based
 4924 intelligent service. While this type of intelligent service is one of the more mature
 4925 AI-based services available on the market—and is largely popular with develop-
 4926 ers [84]—further evaluations of the our tactic may need to be explored against other
 4927 endpoints (i.e., object localisation) or, indeed, other types of services, such as natural
 4928 language processing, audio transcription, or on time-series data. Future studies may
 4929 need to explore this avenue of research.



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     ↪ ,
17   "reason": "Exceeded confidence delta threshold ±0.01 in 5
17     ↪ labels (delta mean=+0.1174). " } }
```

Figure 10.9: 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 10.10: 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", "iguanidae"
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 10.11: Example of an expected label missing due to evolution.

4930 10.5.3.3 Construct Validity

4931 The evaluation of our experiment was largely conducted under clinical conditions,
4932 and a real-world case study of the design and implementation of our proposed tactic
4933 would be beneficial to learn about possible side-effects from implementing such a
4934 design (e.g., implications to cost etc.). Therefore, our evaluation does not consider
4935 more practical considerations that a real-world, production-grade system may need
4936 to consider.

4937 10.6 Discussion**4938 10.6.1 Implications****4939 10.6.1.1 For cloud vendors**

4940 Cloud vendors that provide IWSs may wish to adopt the architectural tactic presented
4941 in this paper by providing a proxy, auxiliary service (or similar) to their existing ser-
4942 vices, thereby improving the current robustness of these services. Further, they
4943 should consider enabling developers of this technical domain knowledge by pre-
4944 venting client applications from using the service without providing a benchmark
4945 dataset, such that the service will return HTTP error codes. These procedures should
4946 be well-documented within the service’s API documentation, thereby indicating to
4947 developers how they can build more robust applications with their IWSs. Lastly,
4948 cloud vendors should consider updating the internal machine learning models less
4949 frequently unless substantial improvements are being made. Many different appli-
4950 cations from many different domains are using these IWSs so it is unlikely that
4951 the model changes are improving all applications. Versioned endpoints would help
4952 with this issue, although—as we have discussed—context using benchmark datasets
4953 should be provided.

4954 10.6.1.2 For application developers

4955 Developers need to monitor all IWSs for evolution using a benchmark dataset and
4956 application specific thresholds before diving straight into using them. It is clear that
4957 the evolutionary issues have significant impact in their client applications [81], and
4958 therefore they need to check the extent this evolution has between versions of an
4959 intelligent service (should versioned APIs be available). Lastly, application devel-
4960 opers should leverage the concept of a proxy server (or other form of intermediary)
4961 when using IWSs to make their applications more robust.

4962 10.6.1.3 For project managers

4963 Project managers need to consider the cost of evolution changes on their application
4964 when using IWSs, and therefore should schedule tasks for building maintenance
4965 infrastructure to detect evolution. Consider scheduling tasks that evaluates and
4966 identifies the frequency of evolution for the specific intelligent service being used.

4967 Our research we have found some IWSs that are not versioned but rarely show
4968 behavioural changes due to evolution.

4969 10.6.2 Limitations

4970 In the situation where a solo developer implements the Proxy Service the main
4971 limitation is the cost vs response time trade-off. Developers may want to be notified
4972 as soon as possible when a behavioural change occurs which requires frequent
4973 validation of the Benchmark Dataset. Each time the Benchmark Dataset is validated
4974 each item is sent as a request to the intelligent service. As cloud vendors charge
4975 per request to an intelligent service there are financial implications for operating
4976 the Proxy Service. If the developer optimises for cost then the application will take
4977 longer to respond to the behavioural change potentially impact end users. Developers
4978 need to consider the impact of cost vs response time when using the Proxy Service.

4979 Another limitation of our approach is the development effort required to imple-
4980 ment the Proxy Service. Developers need to build a scheduling component, batch
4981 processing pipeline for the Benchmark Dataset, and a web service. These com-
4982 ponents require developing and testing which impact project schedules and have
4983 maintenance implications. Thus, we advise developers to consider the overhead of
4984 a Proxy Service and way up the benefits with have incorrect behaviour caused by
4985 evolution of IWSs.

4986 10.6.3 Future Work

4987 10.6.3.1 Guidelines to construct and update the Benchmark Dataset

4988 Our approach assumes that each category of evolution is present in the Benchmark
4989 Dataset prepared by the developer. Further guidelines are required to ensure that the
4990 developer knows how to validate the data before using the Proxy Service. Our work
4991 will also need to be extended to support updating the benchmark dataset.

4992 10.6.3.2 Extend the evolution categories to support other IWSs

4993 Further investigation is needed into the evolution characteristics of other IWSs.
4994 The evolution challenges with services that provide optimisation algorithms such as
4995 route planning are likely to differ from CVSs. These characteristics of an applica-
4996 tion domain have shown to greatly influence software architecture [20] and further
4997 development of the Proxy Service will need to account for these differences.

4998 10.6.3.3 Provide tool support for optimising parameters for an application context

4999 Appropriately using the Proxy Service requires careful selection of thresholds,
5000 benchmark rules and schedule. Further work is required to support the developer
5001 in making these decisions so an optimal application specific outcome is achieved.
5002 One approach is a to present the trade-offs to the developer and let them visualise
5003 the impact of their decisions.

5004 10.7 Related Work

5005 10.7.0.1 Robustness of Intelligent Services

5006 While usage of IWSs have been proven to have widespread benefits to the community [88, 272], they are still largely understudied in software engineering literature,
5007 particularly around their robustness in production-grade systems. As an example,
5008 advancements in computer vision (largely due to the resurgence of convolutional
5009 neural networks in the late 1990s [194]) have been made available through IWSs and
5010 are given marketed promises from prominent cloud vendors, e.g., “with Amazon
5011 Rekognition, you don’t have to build, maintain or upgrade deep learning pipelines”.⁵
5012 However, while vendors claim this, the state of the art of *computer vision itself*
5013 is still susceptible to many robustness flaws, as highlighted by many recent studies
5014 [106, 285, 336]. Further, each service has vastly different (and incompatible)
5015 ontologies which are non-static and evolve [81, 245], certain services can mislabel
5016 images when as little as 10% noise is introduced [149], and developers have a shallow
5017 understanding of the fundamental AI concepts behind these issues, which presents a
5018 dichotomy of their understanding of the technical domain when contrasted to more
5019 conventional domains such as mobile application development [84].
5020

5021 10.7.0.2 Proxy Servers as Fault Detectors

5022 Fault detection is an availability tactic that encompasses robustness of software [26].
5023 Our architecture implements the sanity check and condition monitoring techniques
5024 to detect faults [26, 155], by validating the reasonableness of the response from the
5025 intelligent service against the conditions set out in the rules encoded in the benchmark
5026 dataset and behaviour token. As we do in this study, the proxy server pattern can be
5027 used to both detect and action faults in another service as an intermediary between a
5028 client and a server. For example, addressing accessibility issues using proxy servers
5029 has been widely addressed [36, 37, 321, 357] and, more recently, they have been
5030 used to address in-browser JavaScript errors [102].

5031 10.8 Conclusions

5032 IWSs are gaining traction in the developer community, and this is shown with
5033 an evermore growing adoption of CVSSs in applications. These services make
5034 integration of AI-based components far more accessible to developers via simple
5035 RESTful APIs that developers are familiar with, and offer forever-‘improving’ object
5036 localisation and detection models at little cost or effort to developers. However, these
5037 services are dependent on their training datasets and do not return consistent and
5038 deterministic results. To enable robust composition, developers must deal with the
5039 evolving training datasets behind these components and consider how these non-
5040 deterministic components impact their deterministic systems.

⁵<https://aws.amazon.com/rekognition/faqs/>, accessed 21 November 2019.

5041 This paper proposes an integration architectural tactic to deal with these issues
5042 by mapping the evolving and probabilistic nature of these services to deterministic
5043 error codes. We propose a new set of error codes that deal directly with the erroneous
5044 conditions that has been observed in IWSs, such as computer vision. We provide
5045 a reference architecture via a proxy server that returns these errors when they are
5046 identified, and evaluate our architecture, demonstrating its efficacy for supporting
5047 IWS evolution. Further, we provide a labelled dataset of the evolutionary patterns
5048 identified, which was used to evaluate our architecture.

5049

Part III

5050

Postface

CHAPTER 11

5051

5052

5053

Conclusions & Future Work

5054

5055 In this chapter, we provide a summary of the contributions within the body of
5056 this work. We evaluate the significance of the research outcomes to the software
5057 engineering research community and identify potential criticisms of these outcomes.
5058 Lastly, we indicate future avenues of research resulting from this thesis and provide
5059 concluding remarks.

5060 11.1 Contributions of this Work

5061 This thesis has presented three primary contributions to the body of software engi-
5062 neering knowledge. Namely, we have presented an improved understanding in the
5063 landscape of IWSs—concretely, those that provide computer vision—by examining
5064 their runtime behaviour and evolution profile over a longitudinal study (Chapter 4).
5065 The implications of this work emphasise the caution developers need to take be-
5066 fore diving deep into using these services, and highlight the substantial impacts to
5067 software quality if these considerations are ignored. We showed that developers
5068 find working with this software more frustrating when contrasted to conventional
5069 software engineering domains (Chapter 6), and that the distribution of the types of
5070 issues they face differs from that of the types of issues developers face in established
5071 areas such as mobile and web development (Chapter 5). Furthermore, developers
5072 find the completeness of the existing CVS API documentation poor (Chapter 5),
5073 and therefore an investigation into the attributes of what *constitutes* a complete API
5074 document according to literature and how developers respond to the efficacy of these
5075 attributes produced a taxonomy that, when applied to three CVS service providers,
5076 found 12 areas of improvement of the services’ documentation (Chapter 7). This
5077 taxonomy further serves as a go-to ‘checklist’ for any software engineer to review a
5078 prioritised list of documentation elements worth implementing into their own API
5079 documentation. Lastly our investigations into improved intelligent service integra-
5080 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 8 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 9) and prevent evolution evident in CVSs via a client-server intermediary proxy server strategy (Chapter 10). A more extensive discussion into the contributions of this thesis is presented in Section 1.7.

11.1.1 Answers to Research Questions

In this subsection, we directly answer the four primary research questions that were posed in Section 1.4.

11.1.1.1 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 11.1 (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 8) 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

5113 of agreement. As we show in Figure 11.1 (right), **distributions of agreement can**
5114 **be disparate even where services agree on a label for the same image.** Lastly,
5115 while intelligent services that provide computer vision are somewhat stable in the
5116 responses they return, **their responses are non-static.** There is no guarantee that a
5117 request with the same image sent in testing will return the same response, and we
5118 find that this potential evolution risk is not sufficiently communicated to developers.



Figure 11.1: *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%.

5119 11.1.1.2 *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 7 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.*

5120 To understand if these services are sufficiently documented, we first investigated
5121 what constitutes a complete API document, investigating literature and validating
5122 this against developers using a survey. These consist of five dimensions: usage
5123 description (or *how* developers can use the API); design rationale (or *when* the de-
5124 velopers should use it); domain concepts (or *why* developers should use it in their
5125 application domain); support artefacts (or *what* additional documentation could be
5126 provided to support developers); and, documentation presentation (or *visualisation*
5127 of the prior four dimensions). This taxonomy is presented with further detail in Ap-
5128 pendix C. **Developers and literature agree code snippets are the most important**
5129 **documentation artefact, followed closely by tutorials and low-level reference**

5130 **documentation.** When we apply this taxonomy to intelligent services such as
5131 CVSSs, we find that there can be improvements made to all dimensions except docu-
5132 mentation presentation, which is sufficient. **The largest suggested improvements**
5133 **fall into the usage description dimension**, in which quick-start guides, step-by-step
5134 tutorials, reference applications, best-practices, listings of all API components, min-
5135 imum system dependencies, and installation instructions require further detail. The
5136 second largest dimension falls into the domain concepts behind computer vision,
5137 where vendors should provide a greater emphasis behind computer vision concepts
5138 and definitions of relevant computer vision terminology (especially since many ven-
5139 dors refer to the same concept under different terms, such as ‘image tagging’ and
5140 ‘label detection’ for what is essentially object recognition). The lack of complete
5141 documentation in domain concepts was further reflected in developer discussions
5142 on Stack Overflow, as found in Chapter 5. Section 7.6.5 details these suggested
5143 improvements in greater detail.

5144 *11.1.1.3 RQ3: “Are CVSS more misunderstood than conventional software engi-
5145 neering 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.*

5146 As expressed on Stack Overflow, we find developers struggle most with simple de-
5147 buggling issues, which reflects a shallow understanding of the of the AI concepts that
5148 empower these services. **The technical nuances become so abstracted away that**
5149 **developers begin to lack a full appreciation of the context and proper usage of**
5150 **these systems.** These questions reveal how developers do not have a strong grasp
5151 of the behaviour of these services and how further functional capability needs to be
5152 overcome by secondary phases of work, such as pre- and post-processing. **Their**
5153 **conceptual understanding of these services are poor**, with our findings sug-
5154 gesting that developers present a misunderstanding of the vocabulary used within
5155 computer vision, such as the differences between object and facial detection, local-
5156 isation and recognition. The lack of strong conceptual understanding also reflects
5157 in discrepancy-based issues where developers cannot appreciate why services result
5158 in specific outcomes contrary to what they believe should happen. **We find these**
5159 **discrepancy-based issues to be the most frustrating for developers**, and argue that
5160 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 11.2). 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.

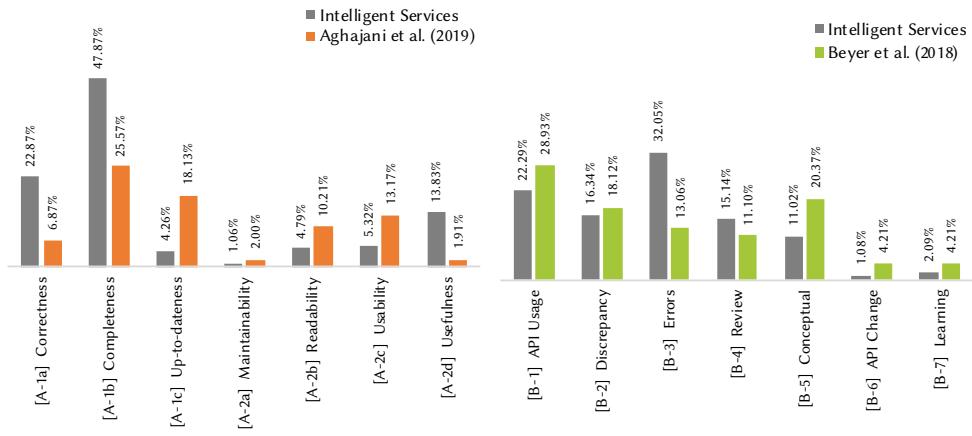


Figure 11.2: The distribution of documentation-specific questions (*left*) and generalised questions (*right*) differs between prior studies. Descriptions of each category for both question types are found in Table 5.1.

11.1.1.4 *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 10 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 dataset 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 8 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 10. 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 9), 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.

11.1.2 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 [277], many other providers of computer vision intelligence exist [360, 376, 377, 378, 384, 397, 398, 400, 449, 450], including those from Asian market [374, 375, 396, 415, 416] 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

5219 thesis strongly emphasises investigations into identifying issues within web-based
5220 intelligent services. We establish a better understanding on their nature and run-time
5221 behaviour (RQ1), how they are documented (RQ2), and how well they are understood
5222 by developers (RQ3), but only offer limited solutions to these issues (e.g., RQ4).
5223 We encourage the software engineering community to use the issues identified in
5224 this work as a stepping-stone into future solutions, identifying other ways (beyond
5225 improved integration techniques) in which developers can handle these issues.

5226 ⟨ *todo: JG: what will you say here?* ⟩

5227 11.2 Concluding Remarks

5228 To our knowledge, little prior investigation has been conducted to understand IWSs
5229 via the lenses of software quality—primarily the robustness, reliability of the services
5230 and completeness of its documentation. In this thesis, we have shown that the non-
5231 deterministic and probabilistic properties of computer vision IWSs present non-
5232 trivial impacts to the quality of software that they are integrated with, and it is
5233 pivotal that developers have a greater appreciation of the technical domain behind
5234 the AI techniques that empower such services.

5235 In identifying evolutionary and run-time issues of these services, the ways in
5236 which they are (currently) documented and these issues communicated (or not!), and
5237 analysing how developers perceive these services with a deterministic mindset, we
5238 have shown just how fragile the use of such services (as they stand) are. We strongly
5239 encourage vendors to use suggestions made within this research to improve both
5240 their documentation and their integration strategies so that developers can ensure
5241 more robust applications when using these services. Ultimately, intelligent AI
5242 components are still in a nascent stage, and therefore strongly suggest one message
5243 to eager developers: use with caution and be aware of the consequences!

5244

5245

5246

References

- 5247 [1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving,
5248 M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker,
5249 V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: A system for large-
5250 scale machine learning,” in *Proceedings of the 12th USENIX Symposium on Operating Systems
Design and Implementation*. Savannah, GA, USA: ACM, 2016. ISBN 978-1-93-197133-1 pp.
5251 265–283.
- 5252 [2] E. Aghajani, C. Nagy, G. Bavota, and M. Lanza, “A Large-scale empirical study on linguistic
5253 antipatterns affecting apis,” in *Proceedings of the 34th International Conference on Software
5254 Maintenance and Evolution*. Madrid, Spain: IEEE, September 2018. DOI 10.1109/IC-
5255 SME.2018.00012. ISBN 978-1-53-867870-1 pp. 25–35.
- 5256 [3] E. Aghajani, C. Nagy, O. L. Vega-Marquez, M. Linares-Vasquez, L. Moreno, G. Bavota,
5257 and M. Lanza, “Software Documentation Issues Unveiled,” in *Proceedings of the 41st Interna-
5258 tional Conference on Software Engineering*. Montreal, QC, Canada: IEEE, May 2019.
5259 DOI 10.1109/ICSE.2019.00122. ISBN 978-1-72-810869-8. ISSN 0270-5257 pp. 1199–1210.
- 5260 [4] M. Ahsanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, “Classifying stack
5261 overflow posts on API issues,” in *Proceedings of the 25th International Conference on
5262 Software Analysis, Evolution and Reengineering*. Campobasso, Italy: IEEE, March 2018.
5263 DOI 10.1109/SANER.2018.8330213. ISBN 978-1-53-864969-5 pp. 244–254.
- 5264 [5] R. E. Al-Qutaish, “Quality Models in Software Engineering Literature: An Analytical and
5265 Comparative Study,” *Journal of American Science*, vol. 6, no. 3, pp. 166–175, 2010.
- 5266 [6] H. Allahyari and N. Lavesson, “User-oriented assessment of classification model under-
5267 standability,” in *Proceedings of the 11th Scandinavian Conference on Artificial Intelligence*, vol.
5268 227. Trondheim, Norway: IOS Press, May 2011. DOI 10.3233/978-1-60750-754-3-11.
5269 ISBN 978-1-60-750753-6. ISSN 0922-6389 pp. 11–19.
- 5270 [7] M. Allamanis and C. Sutton, “Why, when, and what: Analyzing stack overflow questions
5271 by topic, type, and code,” in *Proceedings of the 10th IEEE International Working Con-
5272 ference on Mining Software Repositories*. San Francisco, CA, USA: IEEE, May 2013.
5273 DOI 10.1109/MSR.2013.6624004. ISBN 978-1-46-732936-1. ISSN 2160-1852 pp. 53–56.
- 5274 [8] J. Alway and C. Calhoun, *Critical Social Theory: Culture, History, and the Challenge of
5275 Difference*. American Sociological Association, 1997, vol. 26, no. 1, DOI 10.2307/2076647.
- 5276 [9] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh, “Modeltracker:
5277 Redesigning performance analysis tools for machine learning,” in *Proceedings of the 33rd
5278 Annual ACM Conference on Human Factors in Computing Systems*. Seoul, Republic of
5279 Korea: ACM, April 2015. DOI 10.1145/2702123.2702509. ISBN 978-1-45-033145-6 pp.
5280 337–346.
- 5281 [10] K. Arnold, “Programmers are People, Too,” *ACM Queue*, vol. 3, no. 5, pp. 54–59, 2005,
5282 DOI 10.1145/1071713.1071731. ISSN 1542-7749

- [11] 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.
- [12] W. R. Ashby and J. R. Pierce, "An Introduction to Cybernetics," *Physics Today*, vol. 10, no. 7, pp. 34–36, July 1957.
- [13] 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
- [14] 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
- [15] 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.
- [16] 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.
- [17] 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.
- [18] K. Ballinger, "Simplicity and Utility, or, Why SOAP Lost," [Online] Available: <http://bit.ly/37vLms0>, December 2014, Accessed: 28 August 2018.
- [19] 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
- [20] S. Barnett, "Extracting technical domain knowledge to improve software architecture," Ph.D. dissertation, Swinburne University of Technology, Hawthorn, VIC, Australia, 2018.
- [21] 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.
- [22] 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.
- [23] 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
- [24] 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
- [25] 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
- [26] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley, 2003. ISBN 0-32-115495-9
- [27] 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. Gargoya, 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,

- 5339 J. Annuscheit, P. Hufnagl, M. Valkonen, K. Kartasalo, L. Latonen, P. Ruusuvuori, K. Li-
5340 imatainen, S. Albarqouni, B. Mungal, A. George, S. Demirci, N. Navab, S. Watanabe, S. Seno,
5341 Y. Takenaka, H. Matsuda, H. A. Phoulady, V. Kovalev, A. Kalinovsky, V. Liauchuk, G. Bueno,
5342 M. M. Fernandez-Carrobles, I. Serrano, O. Deniz, D. Racoceanu, and R. Venâncio, "Diagnostic
5343 assessment of deep learning algorithms for detection of lymph node metastases in women with
5344 breast cancer," *Journal of the American Medical Association*, vol. 318, no. 22, pp. 2199–2210,
5345 December 2017, DOI 10.1001/jama.2017.14585. ISSN 1538-3598
- 5346 [28] R. Bellazzi and B. Zupan, "Predictive data mining in clinical medicine: Current issues and
5347 guidelines," *International Journal of Medical Informatics*, vol. 77, no. 2, pp. 81–97, 2008,
5348 DOI 10.1016/j.ijmedinf.2006.11.006. ISSN 1386-5056
- 5349 [29] A. Ben-David, "Monotonicity Maintenance in Information-Theoretic Machine Learning Algo-
5350 rithms," *Machine Learning*, vol. 19, no. 1, pp. 29–43, 1995, DOI 10.1023/A:1022655006810.
5351 ISSN 1573-0565
- 5352 [30] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifier (URI): Generic
5353 syntax," Tech. Rep., 2004.
- 5354 [31] L. L. Berry, A. Parasuraman, and V. A. Zeithaml, "SERVQUAL: A multiple-item scale for
5355 measuring consumer perceptions of service quality," *Journal of Retailing*, vol. 64, no. 1, pp.
5356 12–40, 1988, DOI 10.1016/S0148-2963(99)00084-3. ISBN 00224359. ISSN 0022-4359
- 5357 [32] J. Bessin, "The Business Value of Quality," [Online] Available: <https://ibm.co/2u0UDK0>, June
5358 2004.
- 5359 [33] S. Beyer and M. Pinzger, "A manual categorization of android app development issues on stack
5360 overflow," in *Proceedings of the 30th International Conference on Software Maintenance and
5361 Evolution*. Victoria, BC, Canada: IEEE, September 2014. DOI 10.1109/ICSME.2014.88.
5362 ISBN 978-0-76-955303-0 pp. 531–535.
- 5363 [34] S. Beyer, C. MacHo, M. Pinzger, and M. Di Penta, "Automatically classifying posts into question
5364 categories on stack overflow," in *Proceedings of the 26th International Conference on Program
5365 Comprehension*. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3196321.3196333.
5366 ISBN 978-1-45-035714-2. ISSN 0270-5257 pp. 211–221.
- 5367 [35] J. Biggs and K. Collis, "Evaluating the Quality of Learning: The SOLO Taxonomy (Structure
5368 of the Observed Learning Outcome)," *Management in Education*, vol. 1, no. 4, p. 20, 1987,
5369 DOI 10.1177/089202068700100412. ISBN 0-12-097551-1. ISSN 0892-0206
- 5370 [36] J. P. Bigham, R. S. Kaminsky, R. E. Ladner, O. M. Danielsson, and G. L. Hempton, "WebInSight:
5371 Making web images accessible," in *Proceedings of the 8th International ACM SIGACCESS
5372 Conference on Computers and Accessibility*. Portland, OR, USA: ACM, October 2006.
5373 DOI 10.1145/1168987.1169018, pp. 181–188.
- 5374 [37] J. P. Bigham, C. M. Prince, and R. E. Ladner, "WebAnywhere," in *Proceedings of the 2008
5375 International Cross-Disciplinary Conference on Web Accessibility*. Beijing, China: ACM,
5376 April 2008. DOI 10.1145/1368044.1368060, pp. 73–82.
- 5377 [38] J. J. Blake, L. P. Maguire, T. M. McGinnity, B. Roche, and L. J. McDaid, "The implementation of
5378 fuzzy systems, neural networks and fuzzy neural networks using FPGAs," *Information Sciences*,
5379 vol. 112, no. 1-4, pp. 151–168, 1998, DOI 10.1016/S0020-0255(98)10029-4. ISSN 0020-0255
- 5380 [39] B. S. Bloom, *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*, 2nd ed.
5381 Addison-Wesley Longman, 1956. ISBN 978-0-58-228010-6
- 5382 [40] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in
5383 *Proceedings of the 2nd International Conference on Software Engineering*. San Francisco,
5384 California, USA: IEEE, October 1976. ISSN 0270-5257 pp. 592–605.
- 5385 [41] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Software Management*, pp.
5386 419–421, 2007, DOI 10.1109/9780470049167.ch12. ISBN 978-0-47-004916-7
- 5387 [42] B. W. Boehm, *Software engineering economics*. Englewood Cliffs, NJ, USA: Prentice-Hall,
5388 1981. ISBN 0-13-822122-7
- 5389 [43] S. Borsci, S. Federici, and M. Lauriola, "On the dimensionality of the System Usability Scale:
5390 A test of alternative measurement models," *Cognitive Processing*, 2009, DOI 10.1007/s10339-
5391 009-0268-9. ISSN 16124782
- 5392 [44] C. Bottomley, "What part writer? What part programmer? A survey of practices
5393 and knowledge used in programmer writing," in *Proceedings of the 2005 IEEE Interna-*

- tional Professional Communication Conference. Limerick, Ireland: IEEE, July 2005. DOI 10.1109/IPCC.2005.1494255, pp. 802–812.

[45] P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge*, 3rd ed. Washington, DC, USA: IEEE, 2014. ISBN 978-0-7695-5166-1

[46] M. Boyd and N. Wilson, “Just ask Siri? A pilot study comparing smartphone digital assistants and laptop Google searches for smoking cessation advice,” *PLoS ONE*, vol. 13, no. 3, 2018, DOI 10.1371/journal.pone.0194811. ISSN 1932-6203

[47] O. Boz, “Extracting decision trees from trained neural networks,” in *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton, AB, Canada: ACM, July 2002. DOI 10.1145/775107.775113, pp. 456–461.

[48] H. B. Braiek and F. Khomh, “On Testing Machine Learning Programs,” *arXiv preprint arXiv:1812.02257*, December 2018.

[49] M. Bramer, *Principles of Data Mining*, ser. Undergraduate Topics in Computer Science. London, England, UK: Springer, 2016, vol. 180, DOI 10.1007/978-1-4471-7307-6. ISBN 978-1-44-717306-9

[50] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, “Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing System*. Boston, MA, USA: ACM, April 2009. DOI 10.1145/1518701.1518944. ISBN 978-1-60-558247-4 pp. 1589–1598.

[51] L. Brathall and M. Jørgensen, “Can you trust a single data source exploratory software engineering case study?” *Empirical Software Engineering*, 2002, DOI 10.1023/A:1014866909191. ISSN 1382-3256

[52] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, “What’s your ML Test Score? A rubric for ML production systems,” in *Proceedings of the 30th Annual Conference on Neural Information Processing Systems*. Barcelona, Spain: Curran Associates Inc., December 2016.

[53] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. New York, NY, USA: CRC press, 1984. DOI 10.1201/9781315139470. ISBN 978-1-35-146049-1

[54] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, “Lessons from applying the systematic literature review process within the software engineering domain,” *Journal of Systems and Software*, vol. 80, no. 4, pp. 571–583, April 2007, DOI 10.1016/j.jss.2006.07.009. ISSN 0164-1212

[55] J. Brooke, “SUS-A quick and dirty usability scale,” in *Usability Evaluation in Industry*. Cornwall, England, UK: Taylor & Francis Ltd, 1996, ch. 21, pp. 189–194. ISBN 978-0-74-840460-5

[56] ——, “SUS: a retrospective,” *Journal of Usability Studies*, vol. 8, no. 2, pp. 29–40, 2013. ISSN 1931-3357

[57] O. Bruna, H. Avetisyan, and J. Holub, “Emotion models for textual emotion classification,” *Journal of Physics: Conference Series*, vol. 772, p. 12063, November 2016, DOI 10.1088/1742-6596/772/1/012063.

[58] M. Bunge, “A General Black Box Theory,” *Philosophy of Science*, vol. 30, no. 4, pp. 346–358, October 1963, DOI 10.1086/287954. ISSN 0031-8248

[59] BusinessWire, “FileShadow Delivers Machine Learning to End Users with Google Vision API | Business Wire,” [Online] Available: <https://bwnews.pr/2O5qv78>, July 2018, Accessed: 25 January 2019.

[60] A. Bussone, S. Stumpf, and D. O’Sullivan, “The role of explanations on trust and reliance in clinical decision support systems,” in *Proceedings of the 2015 IEEE International Conference on Healthcare Informatics*. Dallas, TX, USA: IEEE, October 2015. DOI 10.1109/ICHI.2015.26. ISBN 978-1-46-739548-9 pp. 160–169.

[61] F. Calefato, F. Lanobile, and N. Novielli, “EmoTxt: a toolkit for emotion recognition from text,” in *Proceedings of the 7th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos*. San Antonio, TX, USA: IEEE, October 2017. DOI 10.1109/ACIIW.2017.8272591, pp. 79–80.

[62] F. Calefato, F. Lanobile, F. Maiorano, and N. Novielli, “Sentiment polarity detection for software development,” *Empirical Software Engineering*, vol. 23, no. 3, pp. 1352–1382, 2018, DOI 10.1007/s10664-017-9546-9.

- 5449 [63] G. Canfora, "User-side testing of Web Services," in *Proceedings of the 9th European Conference*
5450 *on Software Maintenance and Reengineering*. Manchester, England, UK: IEEE, March 2005.
5451 DOI 10.1109/csmr.2005.57. ISSN 1534-5351 p. 301.
- 5452 [64] G. Canfora and M. Di Penta, "Testing services and service-centric systems: Challenges and
5453 opportunities," *IT Professional*, vol. 8, no. 2, pp. 10–17, 2006, DOI 10.1109/MITP.2006.51.
5454 ISSN 1520-9202
- 5455 [65] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for
5456 healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*,
5457 vol. 2015-Augus. Sydney, Australia: ACM, August 2015. DOI 10.1145/2783258.2788613.
5458 ISBN 978-1-45-033664-2 pp. 1721–1730.
- 5459 [66] F. Casati, H. Kuno, G. Alonso, and V. Machiraju, *Web Services-Concepts, Architectures and
5460 Applications*, 2004. ISBN 978-3-64-207888-0
- 5461 [67] J. P. Cavano and J. A. McCall, "A framework for the measurement of software quality," in
5462 *Proceedings of the Software Quality Assurance Workshop on Functional and Performance
5463 Issues*, vol. 3, no. 5, November 1978, DOI 10.1145/800283.811113, pp. 133–139.
- 5464 [68] C. V. Chambers, D. J. Balaban, B. L. Carlson, and D. M. Grasberger, "The effect of
5465 microcomputer-generated reminders on influenza vaccination rates in a university-based family
5466 practice center." *The Journal of the American Board of Family Practice / American Board of
5467 Family Practice*, vol. 4, no. 1, pp. 19–26, 1991, DOI 10.3122/jabfm.4.1.19. ISSN 0893-8652
- 5468 [69] J. Cheng and R. Greiner, "Learning bayesian belief network classifiers: Algorithms and system,"
5469 in *Proceedings of the 14th Biennial Conference of the Canadian Society for Computational
5470 Studies of Intelligence*, vol. 2056. Ottawa, ON, Canada: Springer, June 2001. DOI 10.1007/3-
5471 540-45153-6_14. ISBN 3-54-042144-0. ISSN 1611-3349 pp. 141–151.
- 5472 [70] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, B. K. Ray, and D. S. Moebus, "Or-
5473 thogonal Defect Classification—A Concept for In-Process Measurements," *IEEE Transactions
5474 on Software Engineering*, vol. 18, no. 11, pp. 943–956, 1992, DOI 10.1109/32.177364. ISSN
5475 0098-5589
- 5476 [71] E. Choi, N. Yoshida, R. G. Kula, and K. Inoue, "What do practitioners ask about code clone? a
5477 preliminary investigation of stack overflow," in *Proceedings of the 9th International Workshop
5478 on Software Clones*, Montreal, QC, Canada, March 2015, DOI 10.1109/IWSC.2015.7069890.
5479 ISBN 978-1-46-736914-5 pp. 49–50.
- 5480 [72] D. V. Cicchetti, "Guidelines, Criteria, and Rules of Thumb for Evaluating Normed and Stan-
5481 dardized Assessment Instruments in Psychology," *Psychological Assessment*, vol. 6, no. 4, pp.
5482 284–290, 1994, DOI 10.1037/1040-3590.6.4.284. ISSN 10403590
- 5483 [73] Digital, "Case Study: Finding defects earlier yields enormous savings," [Online] Available:
5484 <http://bit.ly/36II2cE>, 2003.
- 5485 [74] P. Clark and R. Boswell, "Rule induction with CN2: Some recent improvements," in *Proceed-
5486 ings of the 1991 European Working Session on Learning*. Porto, Portugal: Springer, March
5487 1991. DOI 10.1007/BFb0017011. ISBN 978-3-54-053816-5. ISSN 1611-3349 pp. 151–163.
- 5488 [75] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological
5489 Measurement*, vol. 20, no. 1, pp. 37–46, 1960, DOI 10.1177/00131644600200104. ISSN
5490 1552-3888
- 5491 [76] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations,"
5492 in *Proceedings of the 10th ACM Conference on Recommender Systems*. Boston, MA, USA:
5493 ACM, September 2016. DOI 10.1145/2959100.2959190. ISBN 978-1-45-034035-9 pp. 191–
5494 198.
- 5495 [77] M. W. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained neural
5496 networks," in *Proceedings of the 8th International Conference on Neural Information Processing
5497 Systems*, vol. 8. Denver, CO, USA: MIT Press, December 1996. ISBN 978-0-26-220107-0 pp.
5498 24–30.
- 5499 [78] J. W. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*,
5500 4th ed. SAGE, 2017. ISBN 860-1-40-429618-5
- 5501 [79] P. B. Crosby, *Quality is free: The art of making quality certain*. McGraw-Hill, 1979. ISBN
5502 978-0-07-014512-2
- 5503

- [5504] [80] A. Cummaudo, R. Vasa, and J. Grundy, "What should I document? A preliminary systematic
[5505] mapping study into API documentation knowledge," in *Proceedings of the 13th International*
[5506] *Symposium on Empirical Software Engineering and Measurement*. Porto de Galinhas, Recife,
[5507] Brazil: IEEE, October 2019. DOI 10.1109/ESEM.2019.8870148. ISBN 978-1-72-812968-6.
[5508] ISSN 1949-3789 pp. 1–6.
- [5509] [81] A. Cummaudo, R. Vasa, J. Grundy, M. Abdelrazek, and A. Cain, "Losing Confidence in
[5510] Quality: Unspoken Evolution of Computer Vision Services," in *Proceedings of the 35th IEEE*
[5511] *International Conference on Software Maintenance and Evolution*. Cleveland, OH, USA:
[5512] IEEE, December 2019. DOI 10.1109/ICSME.2019.00051. ISBN 978-1-72-813094-1 pp.
[5513] 333–342.
- [5514] [82] A. Cummaudo, S. Barnett, R. Vasa, and J. Grundy, "Threshy: Supporting Safe Usage of
[5515] Intelligent Web Services," 2020, Unpublished.
- [5516] [83] A. Cummaudo, S. Barnett, R. Vasa, J. Grundy, and M. Abdelrazek, "Beware the evolving
[5517] 'intelligent' web service! An integration architecture tactic to guard AI-first components,"
[5518] 2020, Unpublished.
- [5519] [84] A. Cummaudo, R. Vasa, S. Barnett, J. Grundy, and M. Abdelrazek, "Interpreting Cloud Com-
[5520] puter Vision Pain-Points: A Mining Study of Stack Overflow," in *Proceedings of the 42nd*
[5521] *International Conference on Software Engineering*. Seoul, Republic of Korea: IEEE, October
[5522] 2020, In Press.
- [5523] [85] A. Cummaudo, R. Vasa, and J. Grundy, "Assessing API documentation knowledge for computer
[5524] vision services," 2020, Unpublished.
- [5525] [86] M. K. Curumsing, A. Cummaudo, U. M. Graestch, S. Barnett, and R. Vasa, "Ranking Computer
[5526] Vision Service Issues using Emotion," 2020, Unpublished.
- [5527] [87] M. K. Curumsing, "Emotion-Oriented Requirements Engineering," Ph.D. dissertation, Swin-
[5528] burne University of Technology, Hawthorn, VIC, Australia, 2017.
- [5529] [88] H. da Mota Silveira and L. C. Martini, "How the New Approaches on Cloud Computer Vision
[5530] can Contribute to Growth of Assistive Technologies to Visually Impaired in the Following
[5531] Years?" *Journal of Information Systems Engineering & Management*, vol. 2, no. 2, pp. 1–3,
[5532] 2017, DOI 10.20897/jisem.201709. ISSN 2468-4376
- [5533] [89] R. M. Davison, M. G. Martinsons, and N. Kock, "Principles of canonical action re-
[5534] search," *Information Systems Journal*, vol. 14, no. 1, pp. 65–86, 2004, DOI 10.1111/j.1365-
[5535] 2575.2004.00162.x. ISSN 1350-1917
- [5536] [90] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic
[5537] algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp.
[5538] 182–197, April 2002, DOI 10.1109/4235.996017. ISSN 1089778X
- [5539] [91] K. Dejaeger, F. Goethals, A. Giangreco, L. Mola, and B. Baesens, "Gaining insight into student
[5540] satisfaction using comprehensible data mining techniques," *European Journal of Operational*
[5541] *Research*, vol. 218, no. 2, pp. 548–562, 2012, DOI 10.1016/j.ejor.2011.11.022. ISSN 0377-2217
- [5542] [92] I. Dey, *Qualitative Data Analysis: A User-Friendly Guide for Social Scientists*. New York,
[5543] NY: Routledge, 1993. DOI 10.4324/9780203412497. ISBN 978-0-41-505852-0
- [5544] [93] V. Dhar, D. Chou, and F. Provost, "Discovering interesting patterns for investment decision
[5545] making with GLOWER - A genetic learner overlaid with entropy reduction," *Data Mining and*
[5546] *Knowledge Discovery*, vol. 4, no. 4, pp. 69–80, 2000, DOI 10.1023/A:1009848126475. ISSN
[5547] 1384-5810
- [5548] [94] V. Dibia, A. Cox, and J. Weisz, "Designing for Democratization: Introducing Novices to
[5549] Artificial Intelligence Via Maker Kits," in *Proceedings of the 2017 CHI Conference Extended*
[5550] *Abstracts on Human Factors in Computing Systems*. Denver, CO, USA: ACM, May 2017, pp.
[5551] 381–384.
- [5552] [95] M. Doderer, K. Yoon, J. Salinas, and S. Kwek, "Protein subcellular localization prediction
[5553] using a hybrid of similarity search and Error-Correcting Output Code techniques that produces
[5554] interpretable results," *In Silico Biology*, vol. 6, no. 5, pp. 419–433, 2006. ISSN 1386-6338
- [5555] [96] P. Domingos, "Occam's Two Razors: The Sharp and the Blunt," in *Proceedings of the 4th*
[5556] *International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA:
[5557] AAAI, August 1998. DOI 10.1.1.40.3278, pp. 37–43.
- [5558] [97] B. Dorn and M. Guzdial, "Learning on the job: Characterizing the programming knowl-
[5559] edge and learning strategies of web designers," in *Proceedings of the 28th ACM Conference*

- 5560 *on Human Factors in Computing Systems*, vol. 2. Atlanta, GA, USA: ACM, April 2010.
5561 DOI 10.1145/1753326.1753430. ISBN 978-1-60-558929-9 pp. 703–712.
- 5562 [98] F. Doshi-Velez and B. Kim, “Towards A Rigorous Science of Interpretable Machine Learning,”
5563 *arXiv preprint arXiv:1702.08608*, 2017.
- 5564 [99] F. Doshi-Velez, M. Kortz, R. Budish, C. Bavitz, S. J. Gershman, D. O’Brien, S. Shieber,
5565 J. Waldo, D. Weinberger, and A. Wood, “Accountability of AI Under the Law: The Role of
5566 Explanation,” *SSRN Electronic Journal*, November 2017, In Press, DOI 10.2139/ssrn.3064761.
- 5567 [100] R. G. Dromey, “A model for software product quality,” *IEEE Transactions on Software Engi-
5568 neering*, vol. 21, no. 2, pp. 146–162, 1995, DOI 10.1109/32.345830. ISBN 978-1-11-815666-7.
5569 ISSN 0098-5589
- 5570 [101] C. Drummond and R. C. Holte, “Cost curves: An improved method for visualizing classifier per-
5571 formance,” *Machine Learning*, vol. 65, no. 1, pp. 95–130, October 2006, DOI 10.1007/s10994-
5572 006-8199-5. ISSN 0885-6125
- 5573 [102] T. Durieux, Y. Hamadi, and M. Monperrus, “Fully Automated HTML and Javascript Rewrit-
5574 ing for Constructing a Self-Healing Web Proxy,” in *Proceedings of the 29th International
5575 Symposium on Software Reliability Engineering*. Memphis, TN, USA: IEEE, October 2018.
5576 DOI 10.1109/ISSRE.2018.00012. ISSN 1071-9458 pp. 1–12.
- 5577 [103] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, “Selecting empirical methods for
5578 software engineering research,” in *Guide to Advanced Empirical Software Engineering*, F. Shull,
5579 J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 11, pp. 285–311. ISBN
5580 978-1-84-800043-8
- 5581 [104] W. Elazmeh, S. Matwin, D. O’Sullivan, W. Michalowski, and K. Farion, “Insights from pre-
5582 dicting pediatric asthma exacerbations from retrospective clinical data,” in *Proceedings of the
5583 22nd Conference on Artificial Intelligence*, vol. WS-07-05. Vancouver, BC, Canada: AAAI,
5584 July 2007. ISBN 978-1-57-735332-4 pp. 10–15.
- 5585 [105] N. Elgendi and A. Elragal, “Big data analytics: A literature review paper,” in *Proceedings of
5586 the 10th Industrial Conference on Data Mining*. St. Petersburg, Russia: Springer, July 2014.
5587 DOI 10.1007/978-3-319-08976-8_16. ISSN 1611-3349 pp. 214–227.
- 5588 [106] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno,
5589 and D. Song, “Robust Physical-World Attacks on Deep Learning Visual Classification,” in
5590 *Proceedings of the 2017 IEEE Computer Society Conference on Computer Vision and Pattern
5591 Recognition*, Honolulu, HI, USA, July 2018, DOI 10.1109/CVPR.2018.00175. ISBN 978-1-
5592 53-866420-9. ISSN 1063-6919 pp. 1625–1634.
- 5593 [107] F. Elder, D. Michie, D. J. Spiegelhalter, and C. C. Taylor, “Machine Learning, Neural, and
5594 Statistical Classification.” *Journal of the American Statistical Association*, vol. 91, no. 433, pp.
5595 436–438, 1996, DOI 10.2307/2291432. ISBN 978-0-13-106360-0. ISSN 0162-1459
- 5596 [108] A. J. Feelders, “Prior knowledge in economic applications of data mining,” in *Proceedings of
5597 the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, vol.
5598 1910. Lyon, France: Springer, September 2000. DOI 10.1007/3-540-45372-5_42. ISBN
5599 978-3-54-041066-9. ISSN 1611-3349 pp. 395–400.
- 5600 [109] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,”
5601 Ph.D. dissertation, University of California, Irvine, 2000.
- 5602 [110] I. Finalyson, “Nondeterministic Finite Automata,” [Online] Available: <http://bit.ly/319GOF9>,
5603 Fredericksburg, VA, USA, 2018.
- 5604 [111] H. Foster, S. Uchitel, J. Magee, and J. Kramer, “Model-based verification of Web service
5605 compositions,” in *Proceedings of the 18th International Conference on Automated Software
5606 Engineering*. Linz, Austria: IEEE, September 2004. DOI 10.1109/ase.2003.1240303, pp.
5607 152–161.
- 5608 [112] A. A. Freitas, “A critical review of multi-objective optimization in data mining,” *ACM SIGKDD
5609 Explorations Newsletter*, vol. 6, no. 2, p. 77, 2004, DOI 10.1145/1046456.1046467. ISSN 1931-
5610 0145
- 5611 [113] ———, “Comprehensible classification models,” *ACM SIGKDD Explorations Newsletter*, vol. 15,
5612 no. 1, pp. 1–10, March 2014, DOI 10.1145/2594473.2594475. ISSN 1931-0145
- 5613 [114] A. A. Freitas, D. C. Wieser, and R. Apweiler, “On the importance of comprehensible classi-
5614 fication models for protein function prediction,” *IEEE/ACM Transactions on Computational*

- 5615 *Biology and Bioinformatics*, vol. 7, no. 1, pp. 172–182, 2010, DOI 10.1109/TCBB.2008.47.
5616 ISSN 1545-5963
- 5617 [115] B. J. Frey and D. Dueck, “Clustering by passing messages between data points,” *Science*, vol.
5618 315, no. 5814, pp. 972–976, February 2007, DOI 10.1126/science.1136800. ISSN 0036-8075
- 5619 [116] N. Friedman, D. Geiger, and M. Goldszmidt, “Bayesian Network Classifiers,” *Machine Learn-*
5620 *ing*, vol. 29, no. 2-3, pp. 131–163, 1997, DOI 10.1002/9780470400531.eorms0099. ISSN
5621 0885-6125
- 5622 [117] G. Fung, S. Sandilya, and R. B. Rao, “Rule extraction from linear support vector machines,”
5623 *Studies in Computational Intelligence*, vol. 80, no. 1, pp. 83–107, 2009, DOI 10.1007/978-3-
5624 540-75390-2_4.
- 5625 [118] D. Gachechiladze, F. Lanubile, N. Novielli, and A. Serebrenik, “Anger and its direction in
5626 collaborative software development,” in *Proceedings of the 39th International Conference on*
5627 *Software Engineering: New Ideas and Emerging Technologies Results Track*, IEEE. Buenos
5628 Aires, Argentina: IEEE, May 2017. DOI 10.1109/ICSE-NIER.2017.18, pp. 11–14.
- 5629 [119] M. Gamer, J. Lemon, I. Fellows, and P. Singh, “Irr: various coefficients of interrater reliability,”
5630 *R package version 0.83*, 2010.
- 5631 [120] S. K. Garg, S. Versteeg, and R. Buyya, “SMICloud: A framework for comparing and ranking
5632 cloud services,” in *Proceedings of the 4th IEEE International Conference on Utility and Cloud*
5633 *Computing*. Melbourne, Australia: IEEE, December 2011. DOI 10.1109/UCC.2011.36.
5634 ISBN 978-0-76-954592-9 pp. 210–218.
- 5635 [121] V. Garousi and M. Felderer, “Experience-based guidelines for effective and efficient data ex-
5636 traction in systematic reviews in software engineering,” in *Proceedings of the 21st International*
5637 *Conference on Evaluation and Assessment in Software Engineering*, vol. Part F1286. Karl-
5638 skrona, Sweden: ACM, June 2017. DOI 10.1145/3084226.3084238. ISBN 978-1-45-034804-1
5639 pp. 170–179.
- 5640 [122] V. Garousi, M. Felderer, and M. V. Mäntylä, “Guidelines for including grey literature and
5641 conducting multivocal literature reviews in software engineering,” *Information and Software*
5642 *Technology*, vol. 106, pp. 101–121, 2019, DOI 10.1016/j.infsof.2018.09.006. ISSN 0950-5849
- 5643 [123] D. A. Garvin, “What Does ‘Product Quality’ Really Mean?” *MIT Sloan Management Review*,
5644 vol. 26, no. 1, pp. 25–43, 1984. ISSN 0019-848X
- 5645 [124] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumeé, and K. Craw-
5646 ford, “Datasheets for Datasets,” *arXiv preprint arXiv:1803.09010*, 2018.
- 5647 [125] R. S. Geiger, N. Varoquaux, C. Mazel-Cabasse, and C. Holdgraf, “The Types, Roles, and
5648 Practices of Documentation in Data Analytics Open Source Software Libraries: A Collaborative
5649 Ethnography of Documentation Work,” *Computer Supported Cooperative Work: CSCW: An*
5650 *International Journal*, vol. 27, no. 3-6, pp. 767–802, May 2018, DOI 10.1007/s10606-018-
5651 9333-1. ISSN 15737551
- 5652 [126] GeoSpatial World, “Mapillary and Amazon Rekognition collaborate to build a parking solution
5653 for US cities through computer vision,” [Online] Available: <http://bit.ly/36AdRmS>, September
5654 2018, Accessed: 25 January 2019.
- 5655 [127] M. Gethsiyal Augasta and T. Kathirvalavakumar, “Reverse engineering the neural networks
5656 for rule extraction in classification problems,” *Neural Processing Letters*, vol. 35, no. 2, pp.
5657 131–150, 2012, DOI 10.1007/s11063-011-9207-8. ISSN 1370-4621
- 5658 [128] H. L. Gilmore, “Product conformance cost,” *Quality progress*, vol. 7, no. 5, pp. 16–19, 1974.
- 5659 [129] R. L. Glass, I. Vessey, and V. Ramesh, “RESRES: The story behind the paper “Research in
5660 software engineering: An analysis of the literature”,” *Information and Software Technology*,
5661 vol. 51, no. 1, pp. 68–70, 2009, DOI 10.1016/j.infsof.2008.09.015. ISSN 0950-5849
- 5662 [130] M. W. Godfrey and D. M. German, “The past, present, and future of software evolution,” in
5663 *Proceedings of the 2008 Frontiers of Software Maintenance*, Beijing, China, October 2008,
5664 DOI 10.1109/FOSM.2008.4659256. ISBN 978-1-42-442655-3 pp. 129–138.
- 5665 [131] M. W. Godfrey and Q. Tu, “Evolution in open source software: a case study,” in
5666 *Conference on Software Maintenance*. San Jose, CA, USA: IEEE, August 2000.
5667 DOI 10.1109/icsm.2000.883030, pp. 131–142.
- 5668 [132] Google LLC, “Classification: Thresholding | Machine Learning Crash Course,” [Online] Avail-
5669 able: <http://bit.ly/36oMgWb>, 2019, Accessed: 5 February 2020.

- 5670 [133] 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.
5671 ISSN 0164-1212
- 5672 [134] P. D. Grünwald, *The Minimum Description Length Principle*. MIT press, 2019.
5673 DOI 10.7551/mitpress/4643.001.0001.
- 5674 [135] M. J. Hadley and H. Marc, "Web Application Description Language," [Online] Available:
5675 http://bit.ly/2RXRhQ1, August 2009.
- 5676 [136] H. A. Haensle, C. Fink, R. Schneiderbauer, F. Toberer, T. Buhl, A. Blum, A. Kalloo, A. Ben
5677 Hadj Hassen, L. Thomas, A. Enk, L. Uhlmann, C. Alt, M. Arenbergerova, R. Bakos, A. Baltzer,
5678 I. Bertlich, A. Blum, T. Bokor-Billmann, J. Bowling, N. Braghierioli, R. Braun, K. Budern-
5679 Bakhaya, T. Buhl, H. Cabo, L. Cabrijan, N. Cevic, A. Classen, D. Deltgen, C. Fink, I. Georgieva,
5680 L. E. Hakim-Meibodi, S. Hanner, F. Hartmann, J. Hartmann, G. Haus, E. Hoxha, R. Karls,
5681 H. Koga, J. Kreusch, A. Lallas, P. Majenka, A. Marghoob, C. Massone, L. Mekokishvili,
5682 D. Mestel, V. Meyer, A. Neuberger, K. Nielsen, M. Oliviero, R. Pampena, J. Paoli, E. Pawlik,
5683 B. Rao, A. Rendon, T. Russo, A. Sadek, K. Samhaber, R. Schneiderbauer, A. Schweizer,
5684 F. Toberer, L. Trennheuser, L. Vlahova, A. Wald, J. Winkler, P. Wo'lbing, and I. Zalaudek, "Man
5685 against Machine: Diagnostic performance of a deep learning convolutional neural network for
5686 dermoscopic melanoma recognition in comparison to 58 dermatologists," *Annals of Oncology*,
5687 vol. 29, no. 8, pp. 1836–1842, May 2018, DOI 10.1093/annonc/mdy166. ISSN 1569-8041
- 5688 [137] K. A. Hallgren, "Computing Inter-Rater Reliability for Observational Data: An Overview and
5689 Tutorial," *Tutorials in Quantitative Methods for Psychology*, vol. 8, no. 1, pp. 23–34, February
5690 2012, DOI 10.20982/tqmp.08.1.p023. ISSN 1913-4126
- 5691 [138] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in *Pro-
5692 ceedings of the 30th International Conference on Neural Information Processing Systems*.
5693 Barcelona, Spain: Curran Associates Inc., December 2016. DOI 978-1-51-083881-9. ISSN
5694 1049-5258 pp. 3323–3331.
- 5695 [139] S. Haselbock, R. Weinreich, G. Buchgeher, and T. Kriechbaum, "Microservice Design Space
5696 Analysis and Decision Documentation: A Case Study on API Management," in *Proceedings
5697 of the 11th International Conference on Service-Oriented Computing and Applications, SOCA
5698 2018*, Paris, France, November 2019, DOI 10.1109/SOCA.2018.00008, pp. 1–8.
- 5700 [140] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*, 2nd ed., ser.
5701 Data Mining, Inference, and Prediction. Springer, January 2001.
- 5702 [141] B. Hayete and J. R. Bienkowska, "Götrees: Predicting go associations from protein domain
5703 composition using decision trees," in *Proceedings of the Pacific Symposium on Biocomput-
5704 ing 2005, PSB 2005*. Hawaii, USA: World Scientific Publishing Company, January 2005.
5705 DOI 10.1142/9789812702456_0013. ISBN 9-81-256046-7 pp. 127–138.
- 5706 [142] A. Head, C. Sadowski, E. Murphy-Hill, and A. Knight, "When not to comment: Questions and
5707 tradeoffs with API documentation for C++ projects," in *Proceedings of the 40th International
5708 Conference on Software Engineering*, ser. questions and tradeoffs with API documentation for
5709 C++ projects. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3180155.3180176.
5710 ISSN 0270-5257 pp. 643–653.
- 5711 [143] R. Heckel and M. Lohmann, "Towards Contract-based Testing of Web Services," *Elec-
5712 tronic Notes in Theoretical Computer Science*, vol. 116, pp. 145–156, January 2005,
5713 DOI 10.1016/j.entcs.2004.02.073. ISSN 1571-0661
- 5714 [144] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie, "Dependency
5715 networks for inference, collaborative filtering, and data visualization," *Journal of Machine
5716 Learning Research*, vol. 1, no. 1, pp. 49–75, 2001, DOI 10.1162/153244301753344614. ISSN
5717 1532-4435
- 5718 [145] M. Henning, "API design matters," *Communications of the ACM*, vol. 52, no. 5, pp. 46–56,
5719 2009, DOI 10.1145/1506409.1506424. ISSN 0001-0782
- 5720 [146] F. Hohman, A. Head, R. Caruana, R. DeLine, and S. M. Drucker, "Gamut: A design probe
5721 to understand how data scientists understand machine learning models," in *Proceedings of the
5722 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow, Scotland, UK:
5723 ACM, May 2019. DOI 10.1145/3290605.3300809. ISBN 978-1-45-035970-2
- 5724 [147] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual Analytics in Deep Learning: An
5725 Interrogative Survey for the Next Frontiers," *IEEE Transactions on Visualization and Computer*

- 5726 *Graphics*, vol. 25, no. 8, pp. 2674–2693, 2019, DOI 10.1109/TVCG.2018.2843369. ISSN
5727 1941-0506
- 5728 [148] J. W. Horch, *Practical Guide To Software Quality Management*. Artech House, 2003. ISBN
5729 978-1-58-053604-2
- 5730 [149] H. Hosseini, B. Xiao, and R. Poovendran, “Google’s cloud vision API is not robust to noise,” in
5731 *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications*,
5732 vol. 2017-Decem. Cancun, Mexico: IEEE, December 2017. DOI 10.1109/ICMLA.2017.0-
5733 172. ISBN 978-1-53-861417-4 pp. 101–105.
- 5734 [150] D. Hou and L. Mo, “Content categorization of API discussions,” in *Proceedings of the 29th In-*
5735 *ternational Conference on Software Maintenance*. Eindhoven, Netherlands: IEEE, September
5736 2013. DOI 10.1109/ICSM.2013.17, pp. 60–69.
- 5737 [151] C. Howard, “Introducing Google AI,” [Online] Available: <http://bit.ly/2uI6vAr>, May 2018,
5738 Accessed: 28 August 2018.
- 5739 [152] J. Huang and C. X. Ling, “Using AUC and accuracy in evaluating learning algorithms,”
5740 *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005,
5741 DOI 10.1109/TKDE.2005.50. ISSN 1041-4347
- 5742 [153] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, “An empirical evaluation
5743 of the comprehensibility of decision table, tree and rule based predictive models,” *Decision*
5744 *Support Systems*, vol. 51, no. 1, pp. 141–154, April 2011, DOI 10.1016/j.dss.2010.12.003.
5745 ISSN 0167-9236
- 5746 [154] IEEE, “IEEE Standard Glossary of Software Engineering Terminology,” 1990.
- 5747 [155] J. Ingino, *Software Architect’s Handbook: Become a Successful Software Architect by Imple-*
5748 *menting Effective Architecture Concepts*. Birmingham, England, UK: Packt Publishing, Ltd.,
5749 2018. ISBN 978-1-78862-406-0
- 5750 [156] International Organization for Standardization, “ISO25010:2011 - Systems and software engi-
5751 neering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and
5752 software quality models,” 2011.
- 5753 [157] ——, “ISO 8402:1986 Information Technology - Software Product Evaluation - Quality Char-
5754 acteristics and Guidelines for Their Use,” [Online] Available: <http://bit.ly/37SK4HP>, 1986.
- 5755 [158] ——, “ISO 9000:2015 Quality management systems – Fundamentals and vocabulary,” [Online]
5756 Available: <http://bit.ly/37O4oKo>, 2015.
- 5757 [159] ——, “ISO/IEC 9126 Information Technology - Software Product Evaluation - Quality Charac-
5758 teristics and Guidelines for Their Use,” [Online] Available: <http://bit.ly/2tgMHUE>, November
5759 1999.
- 5760 [160] S. Inzunza, R. Juárez-Ramírez, and S. Jiménez, “API Documentation,” in *Proceedings of the*
5761 *6th World Conference on Information Systems and Technologies*. Naples, Italy: Springer,
5762 March 2018. DOI 10.1007/978-3-319-77712-2_22, pp. 229–239.
- 5763 [161] A. Iyengar, “Supporting Data Analytics Applications Which Utilize Cognitive Services,” in
5764 *Proceedings of the 37th International Conference on Distributed Computing Systems*. Atlanta,
5765 GA, USA: IEEE, June 2017. DOI 10.1109/ICDCS.2017.172. ISBN 978-1-53-861791-5 pp.
5766 1856–1864.
- 5767 [162] N. Japkowicz and M. Shah, *Evaluating learning algorithms: A classification perspective*.
5768 Cambridge University Press, 2011, vol. 9780521196, DOI 10.1017/CBO9780511921803. ISBN
5769 978-0-51-192180-3
- 5770 [163] M. W. M. Jaspers, M. Smeulers, H. Vermeulen, and L. W. Peute, “Effects of clinical decision-
5771 support systems on practitioner performance and patient outcomes: A synthesis of high-quality
5772 systematic review findings,” *Journal of the American Medical Informatics Association*, vol. 18,
5773 no. 3, pp. 327–334, 2011, DOI 10.1136/amiajnl-2011-000094. ISSN 1067-5027
- 5774 [164] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and
5775 D. K. Busse, “Improving documentation for eSOA APIs through user studies,” in *Proceedings*
5776 *of the First International Symposium on End User Development*, vol. 5435 LNCS. Siegen,
5777 Germany: Springer, March 2009. DOI 10.1007/978-3-642-00427-8_6. ISSN 0302-9743 pp.
5778 86–105.
- 5779 [165] T. Jiang and A. E. Keating, “AVID: An integrative framework for discovering functional rela-
5780 tionship among proteins,” *BMC Bioinformatics*, vol. 6, no. 1, p. 136, 2005, DOI 10.1186/1471-
5781 2105-6-136. ISSN 1471-2105

- 5782 [166] B. Jimerson and B. Gregory, "Pivotal Cloud Foundry, Google ML, and Spring," [Online]
5783 Available: <http://bit.ly/2RUBIIL>, San Francisco, CA, USA, December 2017.
- 5784 [167] Y. Jin, *Multi-Objective Machine Learning*, ser. Studies in Computational Intelligence. Berlin,
5785 Heidelberg: Springer, 2006. DOI 10.1007/3-540-33019-4. ISBN 978-3-54-030676-4
- 5786 [168] U. Johansson and L. Niklasson, "Evolving decision trees using oracle guides," in *Proceedings
5787 of the 2009 IEEE Symposium on Computational Intelligence and Data Mining*. Nashville,
5788 TN, USA: IEEE, May 2009. DOI 10.1109/CIDM.2009.4938655. ISBN 978-1-42-442765-9
5789 pp. 238–244.
- 5790 [169] M. Jørgensen, T. Dybå, K. Liestøl, and D. I. K. Sjøberg, "Incorrect results in software engineer-
5791 ing experiments: How to improve research practices," *Journal of Systems and Software*, vol.
5792 116, pp. 133–145, 2016, DOI 10.1016/j.jss.2015.03.065. ISSN 0164-1212
- 5793 [170] J. M. Juran, *Juran on Planning for Quality*. New York, NY, USA: The Free Press, 1988. ISBN
5794 978-0-02-916681-9
- 5795 [171] N. Juristo and O. S. Gómez, "Replication of software engineering experiments," in *Proceedings
5796 of the LASER Summer School on Software Engineering*. Elba Island, Italy: Springer, 2011.
5797 DOI 10.1007/978-3-642-25231-0_2. ISBN 978-3-64-225230-3. ISSN 0302-9743 pp. 60–88.
- 5798 [172] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*. Boston, MA,
5799 USA: Springer, March 2001. DOI 10.1007/978-1-4757-3304-4.
- 5800 [173] A. Karwath and R. D. King, "Homology induction: The use of machine learning to improve se-
5801 quence similarity searches," *BMC Bioinformatics*, vol. 3, no. 1, p. 11, 2002, DOI 10.1186/1471-
5802 2105-3-11. ISSN 1471-2105
- 5803 [174] K. A. Kaufman and R. S. Michalski, "Learning from inconsistent and noisy data: The AQ18
5804 approach," in *Proceedings of the 11th European Conference on Principles and Practice of
5805 Knowledge Discovery in Databases*, vol. 1609. Warsaw, Poland: Springer, September 1999.
5806 DOI 10.1007/BFb0095128. ISBN 3-540-65965-X. ISSN 1611-3349 pp. 411–419.
- 5807 [175] D. Kavaler, D. Posnett, C. Gibler, H. Chen, P. Devanbu, and V. Filkov, "Using and asking:
5808 APIs used in the Android market and asked about in StackOverflow," in *Proceedings of the
5809 5th International Conference on Social Informatics*. Kyoto, Japan: Springer, November 2013.
5810 DOI 10.1007/978-3-319-03260-3_35. ISBN 978-3-31-903259-7. ISSN 0302-9743 pp. 405–
5811 418.
- 5812 [176] B. Kim, "Interactive and Interpretable Machine Learning Models for Human Machine Collab-
5813 oration," Ph.D. dissertation, Massachusetts Institute of Technology, 2015.
- 5814 [177] B. Kim, C. Rudin, and J. Shah, "The Bayesian case model: A generative approach for case-
5815 based reasoning and prototype classification," in *Proceedings of the 28th Conference on Neural
5816 Information Processing Systems*, Montreal, QC, Canada, December 2014. ISSN 1049-5258 pp.
5817 1952–1960.
- 5818 [178] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews
5819 in Software Engineering," Software Engineering Group, Keele University and Department of
5820 Computer Science, University of Durham, Keele, UK, Tech. Rep., 2007.
- 5821 [179] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to Advanced
5822 Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer,
5823 November 2007, ch. 3, pp. 63–92. ISBN 978-1-84-800043-8
- 5824 [180] B. A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-Based Software Engineering," in
5825 *Proceedings of the 26th International Conference on Software Engineering*. Edinburgh,
5826 Scotland, UK: IEEE, May 2004. ISBN 978-0-76-952163-3 pp. 273–281.
- 5827 [181] H. K. Klein and M. D. Myers, "A set of principles for conducting and evaluating interpretive
5828 field studies in information systems," *MIS Quarterly: Management Information Systems*, vol. 23,
5829 no. 1, pp. 67–94, 1999, DOI 10.2307/249410. ISSN 0276-7783
- 5830 [182] A. J. Ko and Y. Riche, "The role of conceptual knowledge in API usability," in *Proceedings of
5831 the 2011 IEEE Symposium on Visual Languages and Human Centric Computing*. Pittsburg,
5832 PA, USA: IEEE, September 2011. DOI 10.1109/VLHCC.2011.6070395. ISBN 978-1-45-
5833 771245-6 pp. 173–176.
- 5834 [183] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming
5835 systems," in *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human
5836 Centric Computing*. Rome, Italy: IEEE, September 2004. DOI 10.1109/vlhcc.2004.47.
5837 ISBN 0-78-038696-5 pp. 199–206.

- 5838 [184] 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
- 5839
- 5840 [185] 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
- 5841
- 5842 [186] K. Krippendorff, *Content Analysis*, ser. An Introduction to Its Methodology. SAGE, 1980. ISBN 978-1-50-639566-1
- 5843
- 5844 [187] 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
- 5845
- 5846 [188] 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
- 5847
- 5848 [189] 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.
- 5849
- 5850 [190] G. Laforge, "Machine Intelligence at Google Scale," in *QCon*, London, England, UK, June 2018.
- 5851
- 5852 [191] 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.
- 5853
- 5854 [192] 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 0006341X
- 5855
- 5856 [193] 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
- 5857
- 5858 [194] 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
- 5859
- 5860 [195] 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.
- 5861
- 5862 [196] 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
- 5863
- 5864 [197] 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
- 5865
- 5866 [198] 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
- 5867
- 5868 [199] 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.
- 5869
- 5870 [200] 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.
- 5871
- 5872 [201] 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.
- 5873
- 5874 [202] 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
- 5875
- 5876
- 5877
- 5878
- 5879
- 5880
- 5881
- 5882
- 5883
- 5884
- 5885
- 5886
- 5887
- 5888
- 5889
- 5890
- 5891
- 5892
- 5893

- 5894 [203] M. Litwin, *How to Measure Survey Reliability and Validity*. Thousand Oaks, CA, USA: SAGE,
5895 1995, vol. 7, DOI 10.4135/9781483348957. ISBN 978-0-80-395704-6
- 5896 [204] Y. Liu, T. Kohlberger, M. Norouzi, G. E. Dahl, J. L. Smith, A. Mohtashamian, N. Olson,
5897 L. H. Peng, J. D. Hipp, and M. C. Stumpe, "Artificial Intelligence-Based Breast Cancer Nodal
5898 Metastasis Detection," *Archives of Pathology & Laboratory Medicine*, vol. 143, no. 7, pp.
5899 859–868, July 2017, DOI 10.5858/arpa.2018-0147-OA. ISSN 1543-2165
- 5900 [205] D. Lo Giudice, C. Mines, A. LeClair, R. Curran, and A. Homan, "How AI Will Change
5901 Software Development And Applications," [Online] Available: <http://bit.ly/38RiAlN>, Forrester
5902 Research, Inc., Tech. Rep., November 2016.
- 5903 [206] R. Lori and M. Oded, *Data mining with decision trees*. World Scientific Publishing Company,
5904 2008, vol. 69. ISBN 978-9-81-277171-1
- 5905 [207] R. E. Lyons and W. Vanderkulk, "The Use of Triple-Modular Redundancy to Improve Computer
5906 Reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, April 2010,
5907 DOI 10.1147/rd.62.0200. ISSN 0018-8646
- 5908 [208] W. Maalej and M. P. Robillard, "Patterns of knowledge in API reference documentation," *IEEE
5909 Transactions on Software Engineering*, 2013, DOI 10.1109/TSE.2013.12. ISSN 0098-5589
- 5910 [209] G. Malgieri and G. Comandé, "Why a right to legibility of automated decision-making exists
5911 in the general data protection regulation," *International Data Privacy Law*, vol. 7, no. 4, pp.
5912 243–265, June 2017, DOI 10.1093/idpl/ixp019. ISSN 2044-4001
- 5913 [210] L. Mandel, "Describe REST Web services with WSDL 2.0," [Online] Available: <https://ibm.co/313RoNV>, May 2008, Accessed: 28 August 2018.
- 5914 [211] T. E. Marshall and S. L. Lambert, "Cloud-based intelligent accounting applications: Accounting
5915 task automation using IBM watson cognitive computing," *Journal of Emerging Technologies
5916 in Accounting*, vol. 15, no. 1, pp. 199–215, 2018, DOI 10.2308/jeta-52095. ISSN 1558-7940
- 5918 [212] D. Martens, J. Vanthienen, W. Verbeke, and B. Baesens, "Performance of classification mod-
5919 els from a user perspective," *Decision Support Systems*, vol. 51, no. 4, pp. 782–793, 2011,
5920 DOI 10.1016/j.dss.2011.01.013. ISSN 0167-9236
- 5921 [213] A. I. Martins, A. F. Rosa, A. Queirós, A. Silva, and N. P. Rocha, "European Portuguese
5922 Validation of the System Usability Scale (SUS)," in *Procedia Computer Science*, 2015,
5923 DOI 10.1016/j.procs.2015.09.273. ISSN 18770509
- 5924 [214] P. Mayring, "Mixing Qualitative and Quantitative Methods," in *Mixed Methodology in Psycho-
5925 logical Research*. Sense Publishers, 2007, ch. 6, pp. 27–36. ISBN 978-9-07-787473-8
- 5926 [215] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in Software Quality: Concept and
5927 Definitions of Software Quality," General Electric Company, Griffiss Air Force Base, NY, USA,
5928 Tech. Rep. RADC-TR-77-369, November 1977.
- 5929 [216] J. McCarthy, "Programs with common sense," in *Proceedings of the Symposium on the Mech-
5930 anization of Thought Processes*, Cambridge, MA, USA, 1963, pp. 1–15.
- 5931 [217] B. McGowen, "Machine learning with Google APIs," [Online] Available: <http://bit.ly/3aUQpo2>, January 2019.
- 5933 [218] M. L. McHugh, "Interrater reliability: The kappa statistic," *Biochemia Medica*, vol. 22, no. 3,
5934 pp. 276–282, 2012, DOI 10.11613/bm.2012.031. ISSN 1330-0962
- 5935 [219] S. G. McLellan, A. W. Roesler, J. T. Tempest, and C. I. Spinuzzi, "Building more usable APIs,"
5936 *IEEE Software*, vol. 15, no. 3, pp. 78–86, 1998, DOI 10.1109/52.676963. ISSN 0740-7459
- 5937 [220] L. McLeod and S. G. MacDonell, "Factors that affect software systems development project
5938 outcomes: A survey of research," *ACM Computing Surveys*, vol. 43, no. 4, p. 24, 2011,
5939 DOI 10.1145/1978802.1978803. ISSN 0360-0300
- 5940 [221] J. Meltzoff and H. Cooper, *Critical thinking about research: Psychology and related fields*,
5941 2nd ed. American Psychological Association, 2018. DOI 10.1037/0000052-000.
- 5942 [222] M. Meng, S. Steinhardt, and A. Schubert, "Application programming interface documentation:
5943 What do software developers want?" *Journal of Technical Writing and Communication*, vol. 48,
5944 no. 3, pp. 295–330, August 2018, DOI 10.1177/0047281617721853. ISSN 1541-3780
- 5945 [223] T. Mens and S. Demeyer, *Software Evolution*. Berlin, Heidelberg: Springer, 2008.
5946 DOI 10.1007/978-3-540-76440-3. ISBN 978-3-54-076439-7
- 5947 [224] T. Mens, S. Demeyer, M. Wermelinger, R. Hirschfeld, S. Ducasse, and M. Jazayeri, "Chal-
5948 lenges in software evolution," in *Proceedings of the 8th International Workshop on Principles*

- 5949 *of Software Evolution*, vol. 2005. Lisbon, Portugal: IEEE, September 2005. DOI 10.1109/I-
5950 WPSE.2005.7. ISBN 0-76-952349-8. ISSN 1550-4077 pp. 13–22.
- 5951 [225] A. C. Michalos and H. A. Simon, *The Sciences of the Artificial*. MIT press, 1970, vol. 11,
5952 no. 1, DOI 10.2307/3102825.
- 5953 [226] D. Michie, “Machine learning in the next five years,” in *Proceedings of the 3rd European
5954 Conference on European Working Session on Learning*. Glasgow, Scotland, UK: Pitman
5955 Publishing, Inc., October 1988. ISBN 978-0-27-308800-4 pp. 107–122.
- 5956 [227] G. A. Miller, “WordNet: A Lexical Database for English,” *Communications of the ACM*, vol. 38,
5957 no. 11, pp. 39–41, November 1995, DOI 10.1145/219717.219748. ISSN 1557-7317
- 5958 [228] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D.
5959 Raji, and T. Gebru, “Model cards for model reporting,” in *Proceedings of the 2nd Conference
5960 on Fairness, Accountability, and Transparency*. Atlanta, GA, USA: ACM, January 2019.
5961 DOI 10.1145/3287560.3287596. ISBN 978-1-45-036125-5 pp. 220–229.
- 5962 [229] D. Moody, “The physics of notations: Toward a scientific basis for constructing visual notations
5963 in software engineering,” *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp.
5964 756–779, 2009, DOI 10.1109/TSE.2009.67. ISSN 0098-5589
- 5965 [230] A. Murgia, P. Tourani, B. Adams, and M. Ortú, “Do developers feel emotions? an ex-
5966 ploratory analysis of emotions in software artifacts,” in *Proceedings of the 11th Work-
5967 ing Conference on Mining Software Repositories*. Hyderabad, India: ACM, May 2014.
5968 DOI 10.1145/2597073.2597086, pp. 262–271.
- 5969 [231] C. Murphy and G. Kaiser, “Improving the Dependability of Machine Learning Applications,”
5970 Department of Computer Science, Columbia University, New York, NY, USA, Tech. Rep. MI,
5971 2008.
- 5972 [232] C. Murphy, G. Kaiser, and M. Arias, “An approach to software testing of machine learning
5973 applications,” in *Proceedings of the 19th International Conference on Software Engineering and
5974 Knowledge Engineering*, Boston, MA, USA, July 2007. ISBN 978-1-62-748661-3 pp. 167–172.
- 5975 [233] B. A. Myers, S. Y. Jeong, Y. Xie, J. Beaton, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K.
5976 Busse, “Studying the Documentation of an API for Enterprise Service-Oriented Architecture,”
5977 *Journal of Organizational and End User Computing*, vol. 22, no. 1, pp. 23–51, January 2010,
5978 DOI 10.4018/joeuc.2010101903. ISSN 1546-2234
- 5979 [234] C. Myers, A. Furqan, J. Nebolsky, K. Caro, and J. Zhu, “Patterns for how users overcome
5980 obstacles in Voice User Interfaces,” in *Proceedings of the 2018 CHI Conference on Human
5981 Factors in Computing Systems*, vol. 2018-April. Montreal, QC, Canada: ACM, April 2018.
5982 DOI 10.1145/3173574.3173580. ISBN 978-1-45-035620-6 p. 6.
- 5983 [235] S. Nakajima, “Model-Checking Verification for Reliable Web Service,” in *Proceedings of the
5984 First International Symposium on Cyber World*. Montreal, QC, Canada: IEEE, November
5985 2002. ISBN 978-0-76-951862-6 pp. 378–385.
- 5986 [236] M. Narayanan, E. Chen, J. He, B. Kim, S. Gershman, and F. Doshi-Velez, “How do Humans
5987 Understand Explanations from Machine Learning Systems? An Evaluation of the Human-
5988 Interpretability of Explanation,” *IEEE Transactions on Evolutionary Computation*, 2018, In
5989 Press.
- 5990 [237] S. Narayanan and S. A. McIlraith, “Simulation, verification and automated composition of web
5991 services,” in *Proceedings of the 11th International Conference on World Wide Web*. Honolulu,
5992 HI, USA: ACM, May 2002. DOI 10.1145/511446.511457. ISBN 1-58-113449-5 pp. 77–88.
- 5993 [238] B. J. Nelson, “Remote Procedure Call,” Ph.D. dissertation, Carnegie Mellon University, 1981.
- 5994 [239] H. F. Niemeyer and A. C. Niemeyer, “Apportionment methods,” *Mathematical Social Sciences*,
5995 vol. 56, no. 2, pp. 240–253, 2008. ISSN 0165-4896
- 5996 [240] Y. Nishi, S. Masuda, H. Ogawa, and K. Uetsuki, “A test architecture for machine learn-
5997 ing product,” in *Proceedings of the 11th International Conference on Software Test-
5998 ing, Verification and Validation Workshops*. Västerås, Sweden: IEEE, April 2018.
5999 DOI 10.1109/ICSTW.2018.00060. ISBN 978-1-53-866352-3 pp. 273–278.
- 6000 [241] N. Novielli, F. Calefato, and F. Lanobile, “The challenges of sentiment detection in the social
6001 programmer ecosystem,” in *Proceedings of the 7th International Workshop on Social Software
6002 Engineering*, Bergamo, Italy, August 2015, DOI 10.1145/2804381.2804387. ISBN 978-1-45-
6003 033818-9 pp. 33–40.

- 6004 [242] ——, “A gold standard for emotion annotation in Stack Overflow,” in *Proceedings of the*
6005 *15th International Conference on Mining Software Repositories*, IEEE. Gothenburg, Sweden:
6006 ACM, May 2018. DOI 10.1145/3196398.3196453, pp. 14–17.
- 6007 [243] K. Nybom, A. Ashraf, and I. Porres, “A systematic mapping study on API documenta-
6008 tion generation approaches,” in *Proceedings of the 44th Euromicro Conference on Software*
6009 *Engineering and Advanced Applications*. Prague, Czech Republic: IEEE, August 2018.
6010 DOI 10.1109/SEAA.2018.00081. ISBN 978-1-53-867382-9 pp. 462–469.
- 6011 [244] J. Nykaza, R. Messinger, F. Boehme, C. L. Norman, M. Mace, and M. Gordon, “What program-
6012 mers really want: Results of a needs assessment for SDK documentation,” in *Proceedings of the*
6013 *20th Annual International Conference on Computer Documentation*. Toronto, ON, Canada:
6014 ACM, October 2002. DOI 10.1145/584955.584976, pp. 133–141.
- 6015 [245] T. Ohtake, A. Cummaudo, M. Abdelrazek, R. Vasa, and J. Grundy, “Merging intelligent API
6016 responses using a proportional representation approach,” in *Proceedings of the 19th Interna-
6017 tional Conference on Web Engineering*. Daejeon, Republic of Korea: Springer, June 2019.
6018 DOI 10.1007/978-3-030-19274-7_28. ISBN 978-3-03-019273-0. ISSN 1611-3349 pp. 391–
6019 406.
- 6020 [246] Open Software Foundation, “Part 3: DCE Remote Procedure Call (RPC),” in *OSF DCE*
6021 *application development guide: revision 1.0*. Prentice Hall, December 1991.
- 6022 [247] N. Oreskes, K. Shrader-Frechette, and K. Belitz, “Verification, validation, and confirmation
6023 of numerical models in the earth sciences,” *Science*, vol. 263, no. 5147, pp. 641–646, 1994,
6024 DOI 10.1126/science.263.5147.641. ISSN 0036-8075
- 6025 [248] A. L. M. Ortiz, “Curating Content with Google Machine Learning Application Programming
6026 Interfaces,” in *EIAPortugal*, July 2017.
- 6027 [249] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams,
6028 “The emotional side of software developers in JIRA,” in *Proceedings of the 13th International*
6029 *Conference on Mining Software Repositories*, ACM. Austin, TX, USA: ACM, May 2016.
6030 DOI 10.1145/2901739.2903505, pp. 480–483.
- 6031 [250] F. E. B. Otero and A. A. Freitas, “Improving the interpretability of classification rules discovered
6032 by an ant colony algorithm: Extended results,” in *Evolutionary Computation*, vol. 24, no. 3.
6033 ACM, 2016. DOI 10.1162/EVCO_a_00155. ISSN 1530-9304 pp. 385–409.
- 6034 [251] A. Pal, S. Chang, and J. A. Konstan, “Evolution of experts in question answering communities,”
6035 in *Proceedings of the 6th International AAAI Conference on Weblogs and Social Media*. Dublin,
6036 Ireland: AAAI, June 2012. ISBN 978-1-57-735556-4 pp. 274–281.
- 6037 [252] R. Parekh, “Designing AI at Scale to Power Everyday Life,” in *Proceedings of the 23rd ACM*
6038 *SIGKDD International Conference on Knowledge Discovery and Data Mining*. Halifax, NS,
6039 Canada: ACM, August 2017. DOI 10.1145/3097983.3105815, p. 27.
- 6040 [253] D. L. Parnas and S. A. Vilkomir, “Precise documentation of critical software,” in *Proceedings*
6041 *of 10th IEEE International Symposium on High Assurance Systems Engineering*. Plano, TX,
6042 USA: IEEE, November 2007. DOI 10.1109/HASE.2007.63. ISSN 1530-2059 pp. 237–244.
- 6043 [254] K. Patel, J. Fogarty, J. A. Landay, and B. Harrison, “Investigating statistical machine learn-
6044 ing as a tool for software development,” in *Proceedings of the 26th SIGCHI Conference on*
6045 *Human Factors in Computing Systems*, ser. CHI ’08. Florence, Italy: ACM, April 2008.
6046 DOI 10.1145/1357054.1357160. ISBN 978-1-60-558011-1 pp. 667–676.
- 6047 [255] C. Pautasso, O. Zimmermann, and F. Leymann, “RESTful web services vs. “Big” web services:
6048 Making the right architectural decision,” in *Proceedings of the 17th International Conference*
6049 *on World Wide Web*. Beijing, China: ACM, April 2008. DOI 10.1145/1367497.1367606.
6050 ISBN 978-1-60-558085-2
- 6051 [256] M. Pazzani, “Comprehensible knowledge discovery: gaining insight from data,” in *Proceedings*
6052 *of the First Federal Data Mining Conference and Exposition*, Washington, DC, USA, 1997, pp.
6053 73–82.
- 6054 [257] M. J. Pazzani, S. Mani, and W. R. Shankle, “Acceptance of rules generated by machine learning
6055 among medical experts,” *Methods of Information in Medicine*, vol. 40, no. 5, pp. 380–385,
6056 2001, DOI 10.1055/s-0038-1634196. ISSN 0026-1270
- 6057 [258] J. Pearl, “The seven tools of causal inference, with reflections on machine learning,” *Communi-
6058 cations of the ACM*, vol. 62, no. 3, pp. 54–60, 2019, DOI 10.1145/3241036. ISSN 1557-7317

- [259] 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.
- [260] 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.
- [261] 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.
- [262] H. Pham, *System Software Reliability*, 1st ed. Springer, 2000. ISBN 978-1-84-628295-9
- [263] 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.
- [264] R. M. Pirsig, *Zen and the art of motorcycle maintenance: An inquiry into values*, 1st ed. HarperTorch, 1974. ISBN 9-780-06-058946-2
- [265] 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
- [266] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 8th ed. McGraw-Hill, 2005. ISBN 978-0-07-802212-8
- [267] D. Pyle, *Data Preparation for Data Mining*, 1st ed. Morgan Kaufmann, 1994. ISBN 978-15-5-860529-9
- [268] 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.
- [269] ——, *C4.5: Programs for machine learning*. San Francisco, CA, USA: Morgan Kauffman, 1993. ISBN 978-1-55-860238-0
- [270] 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.
- [271] 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.
- [272] 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
- [273] 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.
- [274] 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.
- [275] 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

- 6115 [276] G. Ridgeway, D. Madigan, T. Richardson, and J. O’Kane, “Interpretable Boosted Naïve Bayes
6116 Classification,” in *Proceedings of the 4th International Conference on Knowledge Discovery
6117 and Data Mining*. New York, NY, USA: AAAI, 1998, pp. 101–104.
- 6118 [277] RightScale Inc., “State of the Cloud Report: DevOps Trends,” Tech. Rep., 2016.
- 6119 [278] G. Ritzer and E. Guba, “The Paradigm Dialog,” *Canadian Journal of Sociology*, vol. 16, no. 4,
6120 p. 446, 1991, DOI 10.2307/3340973. ISSN 0318-6431
- 6121 [279] M. P. Robillard, “What makes APIs hard to learn? Answers from developers,” *IEEE Software*,
6122 vol. 26, no. 6, pp. 27–34, 2009, DOI 10.1109/MS.2009.193. ISSN 0740-7459
- 6123 [280] M. P. Robillard and R. Deline, “A field study of API learning obstacles,” *Empirical Software
6124 Engineering*, vol. 16, no. 6, pp. 703–732, 2011, DOI 10.1007/s10664-010-9150-8. ISSN 1382-
6125 3256
- 6126 [281] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Geros-
6127 all, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd,
6128 and E. Wong, “On-demand developer documentation,” in *Proceedings of the 33rd IEEE In-
6129 ternational Conference on Software Maintenance and Evolution*. Shanghai, China: IEEE,
6130 September 2017. DOI 10.1109/ICSM.2017.817, pp. 479–483.
- 6131 [282] H. Robinson, J. Segal, and H. Sharp, “Ethnographically-informed empirical studies of soft-
6132 ware practice,” *Information and Software Technology*, vol. 49, no. 6, pp. 540–551, 2007,
6133 DOI 10.1016/j.infsof.2007.02.007. ISSN 0950-5849
- 6134 [283] C. Rosen and E. Shihab, “What are mobile developers asking about? A large scale study
6135 using stack overflow,” *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016,
6136 DOI 10.1007/s10664-015-9379-3. ISSN 1573-7616
- 6137 [284] W. Rosenberry, D. Kenney, and G. Fisher, *Understanding DCE*. O’Reilly & Associates, Inc.,
6138 1992. ISBN 978-1-56-592005-7
- 6139 [285] A. Rosenfeld, R. Zemel, and J. K. Tsotsos, “The Elephant in the Room,” *arXiv preprint
6140 arXiv:1808.03305*, 2018.
- 6141 [286] A. S. Ross, M. C. Hughes, and F. Doshi-Velez, “Right for the right reasons: Training differen-
6142 tiable models by constraining their explanations,” in *Proceedings of the 26th International Joint
6143 Conferences on Artificial Intelligence*, Melbourne, Australia, August 2017, DOI 10.24963/ij-
6144 cai.2017/371. ISBN 978-0-99-924110-3. ISSN 1045-0823 pp. 2662–2670.
- 6145 [287] R. J. Rubey and R. D. Hartwick, “Quantitative measurement of program quality,” in *Proceedings
6146 of the 1968 23rd ACM National Conference*. Las Vegas, NV, USA: ACM, August 1968.
6147 DOI 10.1145/800186.810631. ISBN 978-1-45-037486-6 pp. 671–677.
- 6148 [288] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical techniques for analyzing
6149 concurrent and probabilistic systems*, ser. CRM Monograph Series, P. Panangaden and F. van
6150 Breugel, Eds. American Mathematical Society, 2004, vol. 23.
- 6151 [289] M. E. C. Santos, J. Polvi, T. Taketomi, G. Yamamoto, C. Sandor, and H. Kato, “Usability scale
6152 for handheld augmented reality,” in *Proceedings of the ACM Symposium on Virtual Reality
6153 Software and Technology*, VRST, 2014, DOI 10.1145/2671015.2671019. ISBN 9781450332538
- 6154 [290] J. Sauro and J. R. Lewis, “When designing usability questionnaires, does it hurt to be positive?”
6155 in *Proceedings of the 2011 SIGCHI Conference on Human Factors in Computing Systems*,
6156 Vancouver, BC, Canada, May 2011, DOI 10.1145/1978942.1979266, pp. 2215–2223.
- 6157 [291] M. Schwabacher and P. Langley, “Discovering communicable scientific knowledge from spatio-
6158 temporal data,” in *Proceedings of the 18th International Conference on Machine Learning*.
6159 Williamstown, MA, USA: Morgan Kaufmann, June 2001. ISBN 978-1-55-860778-1 pp. 489–
6160 496.
- 6161 [292] A. Schwaighofer and N. D. Lawrence, *Dataset shift in machine learning*, J. Quiñonero-Candela
6162 and M. Sugiyama, Eds. Cambridge, MA, USA: The MIT Press, 2008. ISBN 978-0-26-217005-
6163 5
- 6164 [293] T. A. Schwandt, “Qualitative data analysis: An expanded sourcebook,” *Evaluation and Program
6165 Planning*, vol. 19, no. 1, pp. 106–107, 1996, DOI 10.1016/0149-7189(96)88232-2. ISSN 0149-
6166 7189
- 6167 [294] D. Sculley, M. E. Otey, M. Pohl, B. Spitznagel, J. Hainsworth, and Y. Zhou, “Detecting
6168 adversarial advertisements in the wild,” in *Proceedings of the 17th ACM SIGKDD International
6169 Conference on Knowledge Discovery and Data Mining*, ACM. San Diego, CA, USA: ACM,
6170 August 2011. DOI 10.1145/2020408.2020455, pp. 274–282.

- [295] 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 *Advances in Neural Information Processing Systems*, 2015, pp. 2503–2511.
- [296] C. B. Seaman, "Qualitative methods," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 2, pp. 35–62. ISBN 978-1-84-800043-8
- [297] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," *International Journal of Computer Vision*, pp. 618–626, 2019, DOI 10.1007/s11263-019-01228-7. ISSN 1573-1405
- [298] S. Sen and L. Knight, "A genetic prototype learner," in *Proceedings of the International Joint Conference on Artificial Intelligence*. Montreal, QC, Canada: Morgan Kaufmann, August 1995, pp. 725–733.
- [299] C. E. Shannon and W. Weaver, "The mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948, DOI 10.1002/j.1538-7305.1948.tb01338.x.
- [300] P. Shaver, J. Schwartz, D. Kirson, and C. O'Connor, "Emotion knowledge: Further exploration of a prototype approach," *Journal of Personality and Social Psychology*, vol. 52, no. 6, pp. 1061–1086, 1987, DOI 10.1037/0022-3514.52.6.1061.
- [301] M. Shaw, "Writing good software engineering research papers," in *Proceedings of the 25th International Conference on Software Engineering*. Portland, OR, USA: IEEE, May 2003. ISBN 978-0-76-951877-0 pp. 726–736.
- [302] M. Shepperd, "Replication studies considered harmful," in *Proceedings of the 40th International Conference on Software Engineering*. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3183399.3183423. ISBN 978-1-45-035662-6. ISSN 0270-5257 pp. 73–76.
- [303] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. New York, NY, USA: Chapman and Hall/CRC, 2004. DOI 10.4324/9780203489536.
- [304] L. Si and J. Callan, "A semisupervised learning method to merge search engine results," *ACM Transactions on Information Systems*, vol. 21, no. 4, pp. 457–491, October 2003, DOI 10.1145/944012.944017. ISSN 1046-8188
- [305] J. Singer, S. E. Sim, and T. C. Lethbridge, "Software engineering data collection for field studies," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 1, pp. 9–34. ISBN 978-1-84-800043-8
- [306] S. Singh, M. T. Ribeiro, and C. Guestrin, "Programs as Black-Box Explanations," *arXiv preprint arXiv:1611.07579*, November 2016.
- [307] V. S. Sinha, S. Mani, and M. Gupta, "Exploring activeness of users in QA forums," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. San Francisco, CA, USA: IEEE, May 2013. DOI 10.1109/MSR.2013.6624010. ISBN 978-1-46-732936-1. ISSN 2160-1852 pp. 77–80.
- [308] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing and Management*, vol. 45, no. 4, pp. 427–437, 2009, DOI 10.1016/j.ipm.2009.03.002. ISSN 0306-4573
- [309] I. Sommerville, *Software Engineering*, 9th ed. Boston, MA, USA: Addison-Wesley, 2011. ISBN 978-0-13-703515-1
- [310] P. Spector, *Summated Rating Scale Construction*. Newbury Park, CA, USA: SAGE, 1992. DOI 10.4135/9781412986038. ISBN 978-0-80-394341-4
- [311] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen, "Asking for (and about) permissions used by Android apps," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. San Francisco, CA, USA: IEEE, May 2013. ISBN 978-1-46-732936-1 pp. 31–40.
- [312] J. Su, D. V. Vargas, and K. Sakurai, "One Pixel Attack for Fooling Deep Neural Networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019, DOI 10.1109/TEVC.2019.2890858. ISSN 1941-0026
- [313] G. H. Subramanian, J. Nosek, S. P. Raghunathan, and S. S. Kanitkar, "A comparison of the decision table and tree," *Communications of the ACM*, vol. 35, no. 1, pp. 89–94, 1992, DOI 10.1145/129617.129621. ISSN 1557-7317

- 6226 [314] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live API documentation," in *Proceedings*
6227 *of the 36th International Conference on Software Engineering*. Hyderabad, India: ACM, May
6228 2014. DOI 10.1145/2568225.2568313. ISSN 0270-5257 pp. 643–652.
- 6229 [315] S. Sun, W. Pan, and L. L. Wang, "A Comprehensive Review of Effect Size Reporting and Inter-
6230 preting Practices in Academic Journals in Education and Psychology," *Journal of Educational*
6231 *Psychology*, vol. 102, no. 4, pp. 989–1004, 2010, DOI 10.1037/a0019507. ISSN 0022-0663
- 6232 [316] D. Szafron, P. Lu, R. Greiner, D. S. Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell,
6233 A. Fyshe, and D. Meeuwis, "Proteome Analyst: Custom predictions with explanations in a web-
6234 based tool for high-throughput proteome annotations," *Nucleic Acids Research*, vol. 32, 2004,
6235 DOI 10.1093/nar/gkh485. ISSN 0305-1048
- 6236 [317] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus,
6237 "Intriguing properties of neural networks," in *Proceedings of the 2nd International Conference*
6238 *on Learning Representations*. Banff, AB, Canada: ACM, April 2014.
- 6239 [318] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Archi-
6240 tecture for Computer Vision," in *Proceedings of the 2016 IEEE Computer Society Conference*
6241 *on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA: IEEE, June 2016.
6242 DOI 10.1109/CVPR.2016.308. ISBN 978-1-46-738850-4. ISSN 1063-6919 pp. 2818–2826.
- 6243 [319] 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.
- 6244 [320] A. Tahir, A. Yamashita, S. Licorish, J. Dietrich, and S. Counsell, "Can you tell me if it
6245 smells? A study on how developers discuss code smells and anti-patterns in Stack Overflow," in
6246 *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software*
6247 *Engineering*. Christchurch, New Zealand: ACM, June 2018. DOI 10.1145/3210459.3210466.
6248 ISBN 978-1-45-036403-4 pp. 68–78.
- 6249 [321] H. Takagi and C. Asakawa, "Transcoding proxy for nonvisual Web access," in *Proceedings of*
6250 *the 2000 ACM Conference on Assistive Technologies*. Arlington, VA, USA: ACM, November
6251 2000. DOI 10.1145/354324.354371, pp. 164–171.
- 6252 [322] G. Tassey, *The economic impacts of inadequate infrastructure for software testing*. National
6253 Institute of Standards and Technology, September 2002. DOI 10.1080/10438590500197315.
6254 ISBN 978-0-75-672618-8
- 6255 [323] A. Taulavuori, E. Niemelä, and P. Kallio, "Component documentation - A key issue in software
6256 product lines," *Information and Software Technology*, vol. 46, no. 8, pp. 535–546, June 2004,
6257 DOI 10.1016/j.infsof.2003.10.004. ISSN 0950-5849
- 6258 [324] R. S. Taylor, "Question-Negotiation and Information Seeking in Libraries," *College and Re-
6259 search Libraries*, vol. 29, no. 3, 1968, DOI 10.5860/crl_29_03_178.
- 6260 [325] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolu-
6261 tion using topic models," *Science of Computer Programming*, vol. 80, pp. 457–479, 2014,
6262 DOI 10.1016/j.scico.2012.08.003. ISSN 0167-6423
- 6263 [326] S. Thrun, "Is Learning The n-th Thing Any Easier Than Learning The First?" in *Proceedings*
6264 *of the 8th International Conference on Neural Information Processing Systems*. Denver, CO,
6265 USA: MIT Press, November 1996. ISSN 1049-5258 p. 7.
- 6266 [327] C. Treude, O. Barzilay, and M. A. Storey, "How do programmers ask and answer questions
6267 on the web?" in *Proceedings of the 33rd International Conference on Software Engineering*.
6268 Honolulu, HI, USA: ACM, May 2011. DOI 10.1145/1985793.1985907. ISBN 978-1-45-
6269 030445-0. ISSN 0270-5257 pp. 804–807.
- 6270 [328] G. Uddin and F. Khomh, "Automatic Mining of Opinions Expressed About APIs in
6271 Stack Overflow," *IEEE Transactions on Software Engineering*, February 2019, In Press,
6272 DOI 10.1109/TSE.2019.2900245. ISSN 1939-3520
- 6273 [329] G. Uddin and M. P. Robillard, "How API Documentation Fails," *IEEE Software*, vol. 32, no. 4,
6274 pp. 68–75, June 2015, DOI 10.1109/MS.2014.80. ISSN 0740-7459
- 6275 [330] M. Usman, R. Britto, J. Börstler, and E. Mendes, "Taxonomies in software engineering: A
6276 Systematic mapping study and a revised taxonomy development method," *Information and*
6277 *Software Technology*, vol. 85, pp. 43–59, May 2017, DOI 10.1016/j.infsof.2017.01.006. ISSN
6278 0950-5849
- 6279 [331] A. Van Assche and H. Blockeel, "Seeing the forest through the trees learning a comprehensible
6280 model from a first order ensemble," in *Proceedings of the 17th International Conference on*

- 6282 *Inductive Logic Programming*. Corvallis, OR, USA: Springer, June 2007. DOI 10.1007/978-
6283 3-540-78469-2_26. ISBN 3-540-78468-3. ISSN 0302-9743 pp. 269–279.
- 6284 [332] R. Vasa, "Growth and Change Dynamics in Open Source Software Systems," Ph.D. dissertation,
6285 Swinburne University of Technology, Hawthorn, VIC, Australia, 2010.
- 6286 [333] B. Venners, "Design by Contract: A Conversation with Bertrand Meyer," *Artima Developer*,
6287 2003.
- 6288 [334] W. Verbeke, D. Martens, C. Mues, and B. Baesens, "Building comprehensible customer churn
6289 prediction models with advanced rule induction techniques," *Expert Systems with Applications*,
6290 vol. 38, no. 3, pp. 2354–2364, 2011, DOI 10.1016/j.eswa.2010.08.023. ISSN 0957-4174
- 6291 [335] F. Wachter, Mitterstadt, "EU regulations on algorithmic decision-making and a "right to expla-
6292 nation"," in *Proceedings of the 2016 ICML Workshop on Human Interpretability in Machine*
6293 *Learning*, New York, NY, USA, June 2016, pp. 26–30.
- 6294 [336] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With great training comes great
6295 vulnerability: Practical attacks against transfer learning," in *Proceedings of the 27th USENIX*
6296 *Security Symposium*. Baltimore, MD, USA: USENIX Association, July 2018. ISBN 978-1-
6297 93-913304-5 pp. 1281–1297.
- 6298 [337] K. Wang, *Cloud Computing for Machine Learning and Cognitive Applications: A Machine*
6299 *Learning Approach*. Cambridge, MA, USA: MIT Press, 2017. ISBN 978-0-26-203641-2
- 6300 [338] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in StackOverflow,"
6301 in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. Coimbra, Portugal:
6302 ACM, March 2013. DOI 10.1145/2480362.2480557, pp. 1019–1024.
- 6303 [339] W. Wang and M. W. Godfrey, "Detecting API usage obstacles: A study of iOS and android
6304 developer questions," in *Proceedings of the 10th Working Conference on Mining Software*
6305 *Repositories*. San Francisco, CA, USA: IEEE, May 2013. DOI 10.1109/MSR.2013.6624006.
6306 ISBN 978-1-46-732936-1. ISSN 2160-1852 pp. 61–64.
- 6307 [340] W. Wang, H. Malik, and M. W. Godfrey, "Recommending Posts concerning API Issues in
6308 Developer Q&A Sites," in *Proceedings of the 12th Working Conference on Mining Software*
6309 *Repositories*. Florence, Italy: IEEE, May 2015. DOI 10.1109/MSR.2015.28. ISBN 978-0-
6310 7695-5594-2. ISSN 2160-1860 pp. 224–234.
- 6311 [341] R. Watson, "Development and application of a heuristic to assess trends in API documenta-
6312 tion," in *Proceedings of the 30th ACM International Conference on Design of Communication*.
6313 Seattle, WA, USA: ACM, October 2012. DOI 10.1145/2379057.2379112. ISBN 978-1-45-
6314 031497-8 pp. 295–302.
- 6315 [342] R. Watson, M. Mark Starnes, J. Jeannot-Schroeder, and J. H. Spyridakis, "API documentation
6316 and software community values: A survey of open-source API documentation," in *Proceedings*
6317 *of the 31st ACM International Conference on Design of Communication*. Greenville, SC,
6318 USA: ACM, September 2013. DOI 10.1145/2507065.2507076, pp. 165–174.
- 6319 [343] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform*
6320 *Architecture*. Crawfordsville, IN, USA: Prentice-Hall, 2005. ISBN 0-13-148874-0
- 6321 [344] D. Wettschereck, D. W. Aha, and T. Mohri, "A Review and Empirical Evaluation of Feature
6322 Weighting Methods for a Class of Lazy Learning Algorithms," *Artificial Intelligence Review*,
6323 vol. 11, no. 1-5, pp. 273–314, 1997, DOI 10.1007/978-94-017-2053-3_11. ISSN 0269-2821
- 6324 [345] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viegas, and J. Wilson, "The What-If
6325 Tool: Interactive Probing of Machine Learning Models," *IEEE Transactions on Visualization*
6326 *and Computer Graphics*, vol. 26, no. 1, pp. 56–65, 2019, DOI 10.1109/tvcg.2019.2934619.
6327 ISSN 1077-2626
- 6328 [346] H. Wickham, "A Layered grammar of graphics," *Journal of Computational and Graphical*
6329 *Statistics*, vol. 19, no. 1, pp. 3–28, January 2010, DOI 10.1198/jcgs.2009.07098. ISSN 1061-
6330 8600
- 6331 [347] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classifi-
6332 cation and evaluation criteria: a proposal and a discussion," *Requirements Engineering*, vol. 11,
6333 no. 1, pp. 102–107, March 2006, DOI 10.1007/s00766-005-0021-6. ISSN 0947-3602
- 6334 [348] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning*
6335 *Tools and Techniques*. Morgan Kaufmann, 2016. DOI 10.1016/c2009-0-19715-5. ISBN
6336 978-0-12-804291-5

- 6337 [349] C. Wohlin and A. Aurum, "Towards a decision-making structure for selecting a research design
6338 in empirical software engineering," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1427–
6339 1455, May 2015, DOI 10.1007/s10664-014-9319-7. ISSN 1573-7616
- 6340 [350] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation*
6341 *in Software Engineering*. Berlin, Heidelberg: Springer, 2012. DOI 10.1007/978-3-642-
6342 29044-2. ISBN 978-3-64-229044-2
- 6343 [351] M. L. Wong and K. S. Leung, *Data Mining Using Grammar Based Genetic Programming and*
6344 *Applications*. Springer, 2002. DOI 10.1007/b116131. ISBN 978-0-79-237746-7
- 6345 [352] M. R. Wrobel, "Emotions in the software development process," in *2013 6th International*
6346 *Conference on Human System Interactions (HSI)*. IEEE, 2013, pp. 518–523.
- 6347 [353] X. Yi and K. J. Kochut, "A CP-nets-based design and verification framework for web services
6348 composition," in *Proceedings of the 2004 IEEE International Conference on Web Services*. San
6349 Diego, CA, USA: IEEE, July 2004. DOI 10.1109/icws.2004.1314810. ISBN 0-76-952167-3
6350 pp. 756–760.
- 6351 [354] R. K. Yin, *Case study research and applications: Design and methods*, 6th ed. Los Angeles,
6352 CA, USA: SAGE, 2017. ISBN 978-1-50-633616-9
- 6353 [355] J. Zahálka and F. Železný, "An experimental test of Occam's razor in classification," *Machine*
6354 *Learning*, vol. 82, no. 3, pp. 475–481, 2011, DOI 10.1007/s10994-010-5227-2. ISSN 0885-6125
- 6355 [356] J. Zhang and R. Kasturi, "Extraction of Text Objects in Video Documents: Recent Progress," in
6356 *Proceedings of the 8th International Workshop on Document Analysis Systems*. Nara, Japan:
6357 IEEE, September 2008. DOI 10.1109/das.2008.49, pp. 5–17.
- 6358 [357] X. Zhang, A. S. Ross, A. Caspi, J. Fogarty, and J. O. Wobbrock, "Interaction Proxies for Runtime
6359 Repair and Enhancement of Mobile Application Accessibility," in *Proceedings of the 2017 CHI*
6360 *Conference on Human Factors in Computing Systems*, ser. CHI '17. Denver, CO, USA: ACM,
6361 May 2017. DOI 10.1145/3025453.3025846. ISBN 978-1-4503-4655-9 pp. 6024–6037.
- 6362 [358] B. Zupan, J. Demšar, M. W. Kattan, J. R. Beck, and I. Bratko, "Machine learning for survival
6363 analysis: a case study on recurrence of prostate cancer," *Artificial intelligence in medicine*,
6364 vol. 20, no. 1, pp. 59–75, 2000.
- 6365 [359] M. Zur Muehlen, J. V. Nickerson, and K. D. Swenson, "Developing web services choreography
6366 standards - The case of REST vs. SOAP," *Decision Support Systems*, vol. 40, no. 1, pp. 9–29,
6367 July 2005, DOI 10.1016/j.dss.2004.04.008. ISSN 0167-9236

6368

6369

List of Online Artefacts

6370

6371 The online artefacts listed below have been downloaded and stored on the Deakin
6372 Research Data Store (RDS) for archival purposes at the following location:

6373 RDS29448-Alex-Cummaudo-PhD/datasets/webrefs

- 6374 [360] Affectiva, Inc., “Home - Affectiva : Affectiva,” <http://bit.ly/36sgbMM>, 2018, accessed: 15
6375 October 2018.
- 6376 [361] Amazon Web Services, Inc., “Detecting Labels in an Image,” <https://amzn.to/2TBNTa>, 2018,
6377 accessed: 28 August 2018.
- 6378 [362] ——, “Detecting Objects and Scenes,” <https://amzn.to/2TDed5V>, 2018, accessed: 28 August
6379 2018.
- 6380 [363] ——, “Amazon Rekognition,” <https://amzn.to/2TyT2BL>, 2018, accessed: 13 September 2018.
- 6381 [364] ——, “Aws release notes,” <https://go.aws/2v0RYjr>, 2019, accessed: 18 March 2019.
- 6382 [365] ——, “Actions - amazon rekognition,” <https://amzn.to/392p3dH>, 2019, accessed: 18 March
6383 2019.
- 6384 [366] ——, “Amazon rekognition | aws machine learning blog,” <https://go.aws/37Q7lKc>, 2019, ac-
6385 cessed: 18 March 2019.
- 6386 [367] ——, “Amazon rekognition image,” <https://go.aws/2ubB6qc>, 2019, accessed: 18 March 2019.
- 6387 [368] ——, “Best practices for sensors, input images, and videos - amazon rekognition,” <https://amzn.to/2uZIW00>, 2019, accessed: 18 March 2019.
- 6388 [369] ——, “Exercise 1: Detect objects and scenes in an image (console) - amazon rekognition,” <https://amzn.to/36TkLnm>, 2019, accessed: 18 March 2019.
- 6389 [370] ——, “Java (sdk v1) code samples for amazon rekognition - aws code sample,” <https://amzn.to/2ugTle3>, 2019, accessed: 18 March 2019.
- 6390 [371] ——, “Limits in amazon rekognition - amazon rekognition,” <https://amzn.to/2On6n0h>, 2019,
6391 accessed: 18 March 2019.
- 6392 [372] ——, “Step 1: Set up an aws account and create an iam user - amazon rekognition,” <https://amzn.to/2tqW4kI>, 2019, accessed: 18 March 2019.
- 6393 [373] ——, “Troubleshooting amazon rekognition video - amazon rekognition,” <https://amzn.to/3b763fS>, 2019, accessed: 18 March 2019.
- 6394 [374] Beijing Geling Shentong Information Technology Co., Ltd., “DeepGlint,” <http://bit.ly/2uHHdPS>, 2018, accessed: 3 April 2019.
- 6395 [375] Beijing Kuangshi Technology Co., Ltd., “Megvii,” <http://bit.ly/2WJYFzk>, 2018, accessed: 3
6396 April 2019.

- 6403 [376] Clarifai, Inc., “Enterprise AI Powered Computer Vision Solutions | Clarifai,” <http://bit.ly/2TB3kSa>, 2018, accessed: 13 September 2018.
- 6404 [377] CloudSight, Inc., “Image Recognition API & Visual Search Results | CloudSight AI,” <http://bit.ly/2UmNPCw>, 2018, accessed: 13 September 2018.
- 6405 [378] Cognitec Systems GmbH, “The face recognition company - Cognitec,” <http://bit.ly/38VguBB>, 2018, accessed: 15 October 2018.
- 6406 [379] A. Cummaudo, <http://bit.ly/2KlyhcD>, 2019, accessed: 27 March 2019.
- 6407 [380] ——, <http://bit.ly/2G7saFJ>, 2019, accessed: 27 March 2019.
- 6408 [381] ——, <http://bit.ly/2G5ZEEe>, 2019, accessed: 27 March 2019.
- 6409 [382] ——, “ICSE 2020 Submission #564 Supplementary Materials,” <http://bit.ly/2Z8zOKW>, 2019.
- 6410 [383] ——, <http://bit.ly/2G6ZOeC>, 2019, accessed: 27 March 2019.
- 6411 [384] Deep AI, Inc., “DeepAI: The front page of A.I. | DeepAI,” <http://bit.ly/2TBNYgf>, 2018, accessed: 26 September 2018.
- 6412 [385] Google LLC, “Best practices for enterprise organizations | documentation | google cloud,” <http://bit.ly/2v0RSs5>, 2019, accessed: 18 March 2019.
- 6413 [386] ——, “Detect Labels | Google Cloud Vision API Documentation | Google Cloud,” <http://bit.ly/2TD5kcy>, 2018, accessed: 28 August 2018.
- 6414 [387] ——, “Class EntityAnnotation | Google.Cloud.Vision.V1,” <http://bit.ly/2TD5fpg>, 2018, accessed: 28 August 2018.
- 6415 [388] ——, “Vision API - Image Content Analysis | Cloud Vision API | Google Cloud,” <http://bit.ly/2TD9mBs>, 2018, accessed: 13 September 2018.
- 6416 [389] ——, “Machine learning glossary | google developers,” <http://bit.ly/3b38VdL>, 2019, accessed: 18 March 2019.
- 6417 [390] ——, “Open Images Dataset V4,” <http://bit.ly/2Ry2vvF>, 2019, accessed: 9 November 2018.
- 6418 [391] ——, “Quickstart: Using client libraries | cloud vision api documentation | google cloud,” <http://bit.ly/2RRMQHG>, 2019, accessed: 18 March 2019.
- 6419 [392] ——, “Release notes | cloud vision api documentation | google cloud,” <http://bit.ly/2UipY5J>, 2019, accessed: 18 March 2019.
- 6420 [393] ——, “Sample applications | cloud vision api documentation | google cloud,” <http://bit.ly/2SdoB5r>, 2019, accessed: 18 March 2019.
- 6421 [394] ——, “Tips & tricks | cloud functions documentation | google cloud,” <http://bit.ly/2GZNc8Z>, 2019, accessed: 18 March 2019.
- 6422 [395] ——, “Vision ai | derive image insights via ml | cloud vision api | google cloud,” <http://bit.ly/31nWoNx>, 2019, accessed: 18 March 2019.
- 6423 [396] Guangzhou Tup Network Technology, “TupuTech,” <http://bit.ly/2uF4IsN>, 2018, accessed: 3 April 2019.
- 6424 [397] Imappa Technologies, “Imappa - powerful image recognition APIs for automated categorization & tagging in the cloud and on-premises,” <http://bit.ly/2TxsyRe>, 2018, accessed: 13 September 2018.
- 6425 [398] International Business Machines Corporation, “Watson Visual Recognition - Overview | IBM,” <https://ibm.co/2TBNIO4>, 2018, accessed: 13 September 2018.
- 6426 [399] ——, “Watson Tone Analyzer,” <https://ibm.co/37w3y4A>, 2019, accessed: 25 January 2019.
- 6427 [400] Kairos AR, Inc., “Kairos: Serving Businesses with Face Recognition,” <http://bit.ly/30WHGNs>, 2018, accessed: 15 October 2018.
- 6428 [401] Microsoft Corporation, “azure-sdk-for-java/ImageTag.java,” <http://bit.ly/38IDPWU>, 2018, accessed: 28 August 2018.
- 6429 [402] ——, “Image Processing with the Computer Vision API | Microsoft Azure,” <http://bit.ly/2YqhkS6>, 2018, accessed: 13 September 2018.
- 6430 [403] ——, “How to call the Computer Vision API,” <http://bit.ly/2TD5oJk>, 2018, accessed: 28 August 2018.
- 6431 [404] ——, “What is Computer Vision?” <http://bit.ly/2TDgUnU>, 2018, accessed: 28 August 2018.
- 6432 [405] ——, “Call the computer vision api - azure cognitive services | microsoft docs,” <http://bit.ly/2vHSdjT>, 2019, accessed: 18 March 2019.
- 6433 [406] ——, “Content tags - computer vision - azure cognitive services | microsoft docs,” <http://bit.ly/2vESzHX>, 2019, accessed: 18 March 2019.

- 6458 [407] ——, “Github - azure-samples/cognitive-services-java-computer-vision-tutorial: This tutorial
6459 shows the features of the microsoft cognitive services computer vision rest api.” <http://bit.ly/37N1yoN>, 2019, accessed: 18 March 2019.
- 6460
- 6461 [408] ——, “Improving your classifier - custom vision service - azure cognitive services | microsoft
6462 docs,” <http://bit.ly/37SBkRQ>, 2019, accessed: 18 March 2019.
- 6463 [409] ——, “Quickstart: Computer vision client library for .net - azure cognitive services | microsoft
6464 docs,” <http://bit.ly/2vF3wJC>, 2019, accessed: 18 March 2019.
- 6465 [410] ——, “Release notes - custom vision service - azure cognitive services | microsoft docs,”
6466 <http://bit.ly/2UIPiaW>, 2019, accessed: 18 March 2019.
- 6467 [411] ——, “Sample: Explore an image processing app in c# - azure cognitive services | microsoft
6468 docs,” <http://bit.ly/2u4mPMh>, 2019, accessed: 18 March 2019.
- 6469 [412] ——, “Tutorial: Generate metadata for azure images - azure cognitive services | microsoft
6470 docs,” <http://bit.ly/2RRnARK>, 2019, accessed: 18 March 2019.
- 6471 [413] ——, “Tutorial: Use custom logo detector to recognize azure services - custom vision - azure
6472 cognitive services | microsoft docs,” <http://bit.ly/2RUGwPH>, 2019, accessed: 18 March 2019.
- 6473 [414] ——, “What is computer vision? - computer vision - azure cognitive services | microsoft docs,”
6474 <http://bit.ly/37SomDx>, 2019, accessed: 18 March 2019.
- 6475 [415] SenseTime, “SenseTime,” <http://bit.ly/2WH6Rjf>, 2018, accessed: 3 April 2019.
- 6476 [416] Shanghai Yitu Technology Co., Ltd., “Yitu Technology,” <http://bit.ly/2uGvxgf>, 2018, accessed:
6477 3 April 2019.
- 6478 [417] Stack Overflow User #1008563 ‘samiles’, “AWS Rekognition PHP SDK gives invalid image
6479 encoding error,” <http://bit.ly/31Sgpec>, 2019, accessed: 22 June 2019.
- 6480 [418] Stack Overflow User #10318601 ‘reza naderii’, “google cloud vision category detecting,”
6481 <http://bit.ly/31Uf32t>, 2019, accessed: 22 June 2019.
- 6482 [419] Stack Overflow User #10729564 ‘gabgob’, “Multiple Google Vision OCR requests at once?”
6483 <http://bit.ly/31P09dU>, 2019, accessed: 22 June 2019.
- 6484 [420] Stack Overflow User #1453704 ‘deoptimancode’, “Human body part detection in Android,”
6485 <http://bit.ly/31T5pxd>, 2019, accessed: 22 June 2019.
- 6486 [421] Stack Overflow User #174602 ‘geekyaleks’, “aws Rekognition not initializing on iOS,” <http://bit.ly/31UeqG9>, 2019, accessed: 22 June 2019.
- 6487
- 6488 [422] Stack Overflow User #2251258 ‘James Dorfman’, “All GoogleVision label possibilities?”
6489 <http://bit.ly/31R4FZi>, 2019, accessed: 22 June 2019.
- 6490 [423] Stack Overflow User #2521469 ‘Hillary Sanders’, “Is there a full list of potential labels that
6491 Google’s Vision API will return?” <http://bit.ly/2KNnJSB>, 2019, accessed: 22 June 2019.
- 6492 [424] Stack Overflow User #2604150 ‘user2604150’, “Google Vision Accent Character Set NodeJs,”
6493 <http://bit.ly/31TsVdp>, 2019, accessed: 22 June 2019.
- 6494 [425] Stack Overflow User #3092947 ‘Mark Bench’, “Google Cloud Vision OCR API returning
6495 incorrect values for bounding box/vertices,” <http://bit.ly/31UeZjf>, 2019, accessed: 22 June
6496 2019.
- 6497 [426] Stack Overflow User #3565255 ‘CSquare’, “Vision API topicality and score always the same,”
6498 <http://bit.ly/2TD5As2>, 2019, accessed: 22 June 2019.
- 6499 [427] Stack Overflow User #4748115 ‘Latifa Al-jiffry’, “similar face recognition using google cloud
6500 vision API in android studio,” <http://bit.ly/31WhMZY>, 2019, accessed: 22 June 2019.
- 6501 [428] Stack Overflow User #4852910 ‘Gaurav Mathur’, “Amazon Rekognition Image caption,” <http://bit.ly/31P08qm>, 2019, accessed: 22 June 2019.
- 6502
- 6503 [429] Stack Overflow User #5294761 ‘Eury Pérez Beltré’, “Specify language for response in Google
6504 Cloud Vision API,” <http://bit.ly/31SsUGG>, 2019, accessed: 22 June 2019.
- 6505 [430] Stack Overflow User #549312 ‘GroovyDotCom’, “Image Selection for Training Visual Recog-
6506 nition,” <http://bit.ly/31W8lcw>, 2019, accessed: 22 June 2019.
- 6507 [431] Stack Overflow User #5809351 ‘J.Doe’, “How to confidently validate object detection results
6508 returned from Google Cloud Vision,” <http://bit.ly/31UcCNy>, 2019, accessed: 22 June 2019.
- 6509 [432] Stack Overflow User #5844927 ‘Amit Pawar’, “Google cloud Vision and Clarifai doesn’t Support
6510 tagging for 360 degree images and videos,” <http://bit.ly/31StuEm>, 2019, accessed: 22 June 2019.
- 6511 [433] Stack Overflow User #5924523 ‘Akash Dathan’, “Can i give aspect ratio in Google Vision api?”
6512 <http://bit.ly/2KSJwsp>, 2019, accessed: 22 June 2019.

- 6513 [434] Stack Overflow User #6210900 ‘Mike Grommet’, “Are the Cloud Vision API limits in documentation correct?” <http://bit.ly/31SsNLg>, 2019, accessed: 22 June 2019.
- 6514 [435] 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.
- 6515 [436] Stack Overflow User #6841211 ‘NigelJL’, “Google Cloud Vision - Numbers and Numerals OCR,” <http://bit.ly/31P07mi>, 2019, accessed: 22 June 2019.
- 6516 [437] 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.
- 6517 [438] Stack Overflow User #7187987 ‘tuanars10’, “Adding a local path to Microsoft Face API by Python,” <http://bit.ly/2KLeMt3>, 2019, accessed: 22 June 2019.
- 6518 [439] Stack Overflow User #7219743 ‘Davide Biraghi’, “Google Vision API does not recognize single digits,” <http://bit.ly/31Ws1Nj>, 2019, accessed: 22 June 2019.
- 6519 [440] 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.
- 6520 [441] Stack Overflow User #7604576 ‘Alagappan Narayanan’, “Text extraction - line-by-line,” <http://bit.ly/31Yc21s>, 2019, accessed: 22 June 2019.
- 6521 [442] 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.
- 6522 [443] Stack Overflow User #7896427 ‘David mark’, “Google Api Vision, ““before_request”” error,” <http://bit.ly/31Z27Zt>, 2019, accessed: 22 June 2019.
- 6523 [444] Stack Overflow User #8210103 ‘Cosmin-Ioan Leferman’, “Google Vision API text detection strange behaviour - Javascript,” <http://bit.ly/31Ucyxi>, 2019, accessed: 22 June 2019.
- 6524 [445] 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.
- 6525 [446] 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.
- 6526 [447] 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.
- 6527 [448] Stack Overflow User #9913535 ‘Sahil Mehra’, “Google Vision API: ModuleNotFoundError: module not found ‘google.oauth2’,” <http://bit.ly/31VIZfU>, 2019, accessed: 22 June 2019.
- 6528 [449] Symisc Systems, S.U.A.R.L, “Computer Vision & Media Processing APIs | PixLab,” <http://bit.ly/2Ulkw9K>, 2018, accessed: 13 September 2018.
- 6529 [450] Talkwalker Inc., “Image Recognition - Talkwalker,” <http://bit.ly/2TyT7W5>, 2018, accessed: 13 September 2018.
- 6530 [451] 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 [284]. This framework used the synchronous communication paradigm remote procedure calls (RPCs) first introduced by Nelson [238] that allows procedures to be called in a remote address space as if it were local. Its communication paradigm, DCE/RPC [246], 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 [246]). 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) [66] 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 [26]. 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 [337, 343]). These attributes are included within SOAP headers; thus, QoS aspects are independent from the transport layer and instead exist at the application layer [255]. 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 [255, 359] 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: [18].



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: [18].

```

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 [255]. That is, the HTTP is considered as a transport

protocol solely. In 2000, Fielding [109] 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 [30].
- **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) [210]). Therefore, no RESTful service uses a standardised response document or invocation syntax. While there are proposals, such as WADL [135], 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: [18].

```
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 (p228)** is a reproduction of Lo Giudice et al. [205]’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 (p229)** 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 (p230)** 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 7, and can be used for future avenues of research.
- **Figure A.5 (p231)** 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 [20].
- **Figure A.6 (p232)** 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 (p232)** 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 9) but was removed due to space limitations.
- **Figure A.8 (p233)** illustrates the domain model of Threshy (Chapter 9).
- **Figure A.9 (p233)** illustrates the dynamic model of using Threshy and its interactions between the application, front-end of Threshy and back-end of Threshy (Chapter 9).
- **Figure A.10 (p234)** 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 (p235)** is a class diagram of the reference architecture of the proposed architecture in Chapter 10. The implementation is provided in Appendix B. See Chapter 10 for more.
- **Figure A.12 (p236)** 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 10 for more.

- **Figure A.13 (p237)** 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 10 for more.
- **Figure A.14 (p238)** is a state diagram that illustrates the overall states that exist within the architecture tactic’s workflows. See Chapter 10 for more.
- **Figure A.15 (p239)** is a sequence diagram illustrating how the reference architecture handles evolution in an external service per the implementation provided in Appendix B. See Chapter 10 for more.
- **Figure A.16 (p240)** 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 10 for more.

Figure A.2: A Broad Range of AI-Based Products And Services Is Already Visible. (From [205].)

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

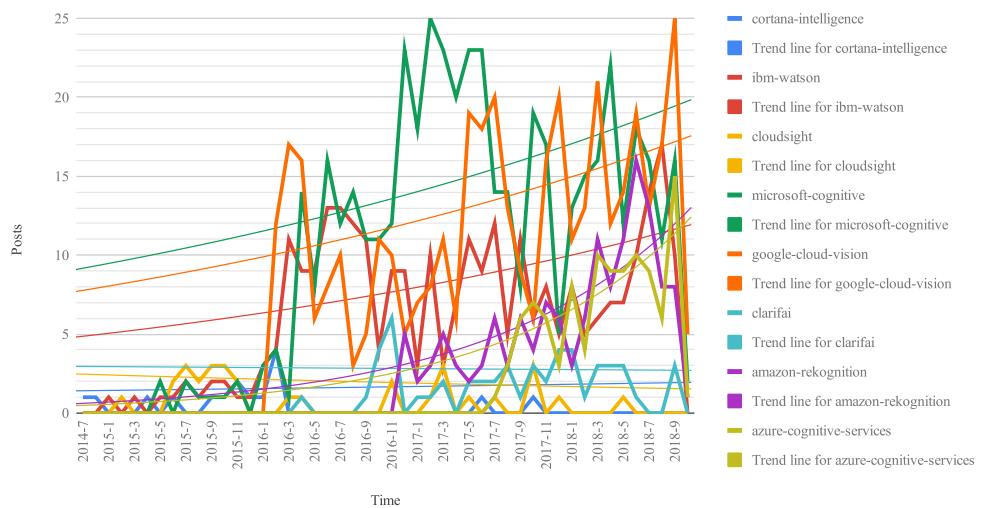
Figure A.3: Increasing interest on Stack Overflow for CVSSs.

Figure A.4: Causal factors that may influence understanding of intelligent web services.

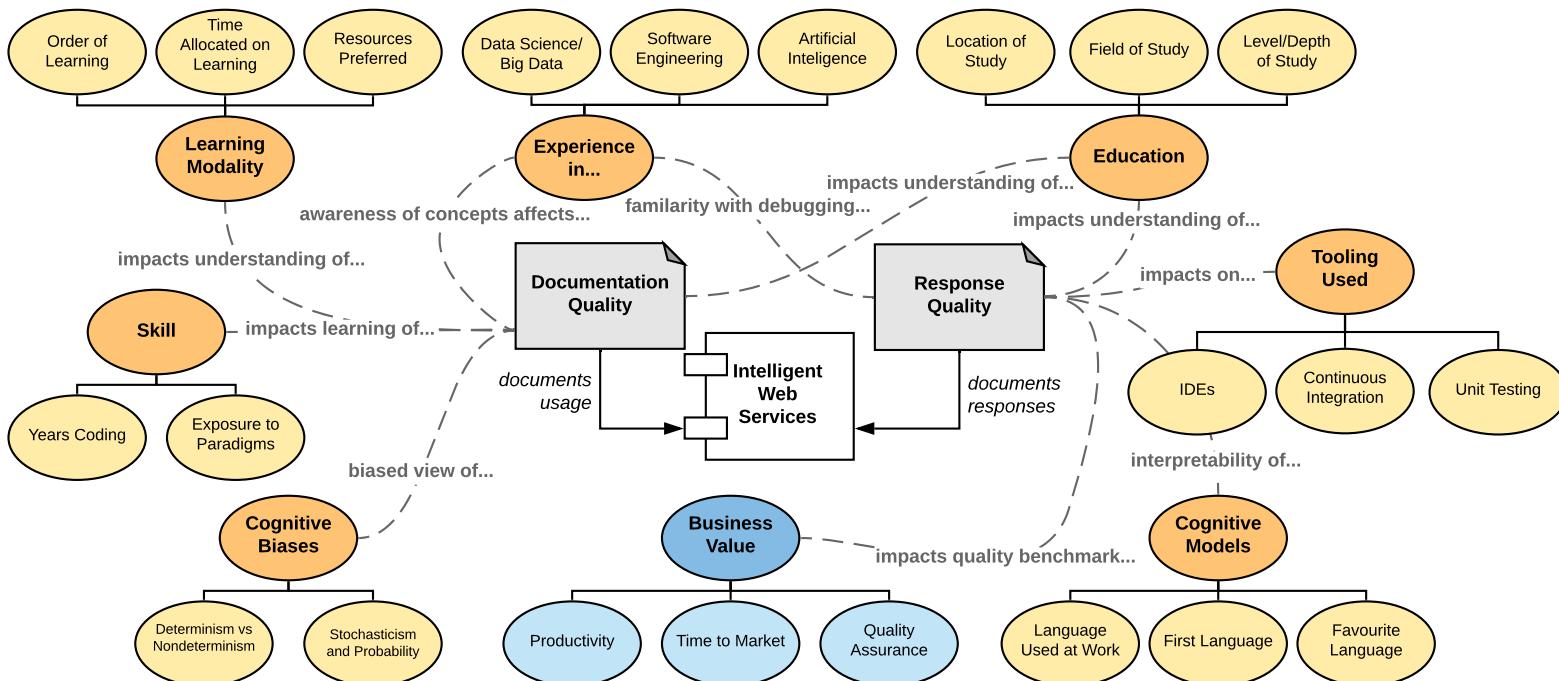


Figure A.5: A proposal technical domain model for intelligent services. (The ⓘ symbol indicates computer vision related services only.)

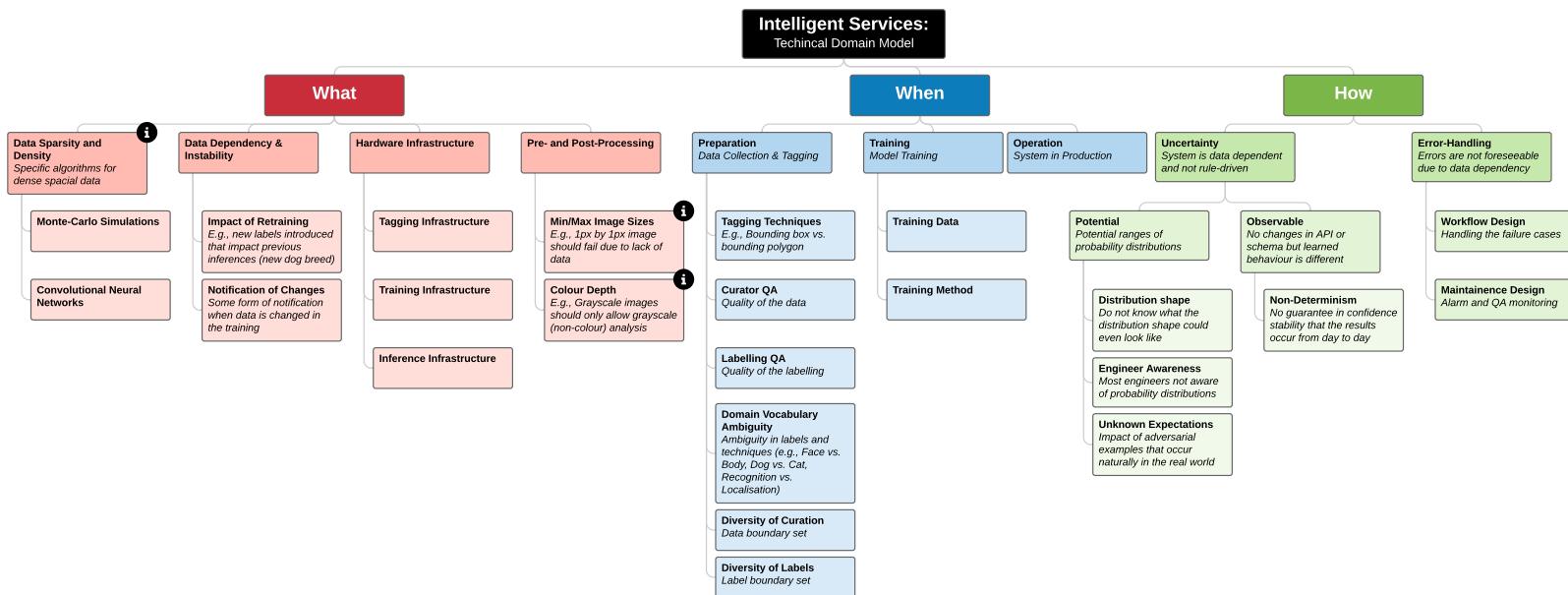


Figure A.6: Potential questions that can be asked around causal factors of a developer's understanding of an intelligent service.

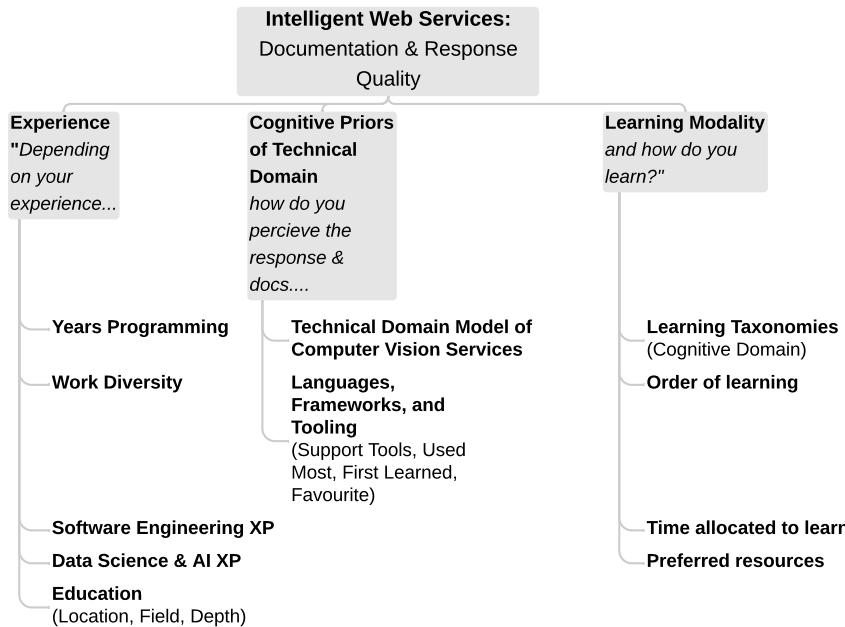


Figure A.7: Threshy assists with making appropriate decision boundaries in the application context by calibrating model (train on an unknown context) to your domain.

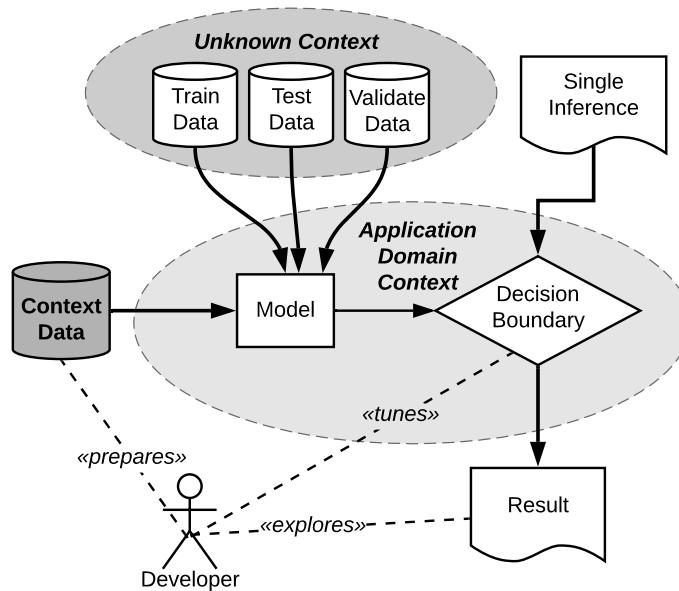


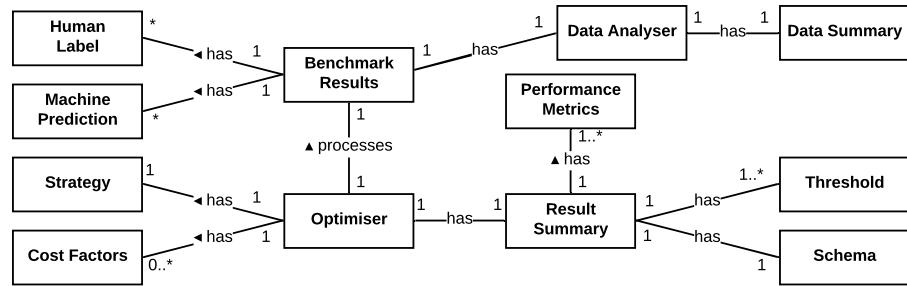
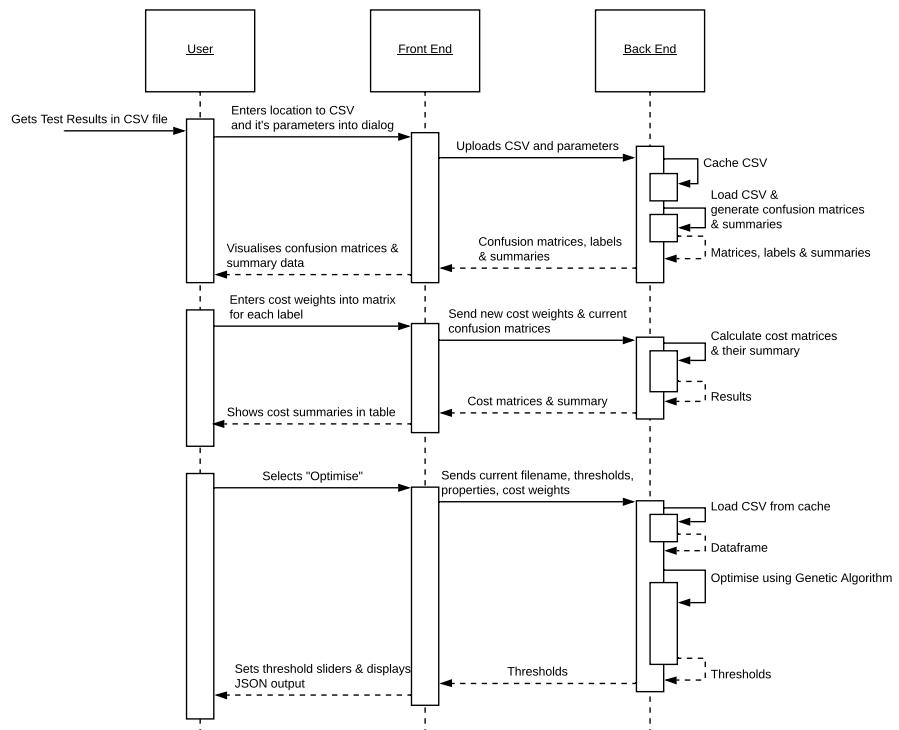
Figure A.8: Threshy domain model.**Figure A.9:** High level overview of Threshy's interaction between the front- and back-end.

Figure A.10: High-level overview of the methodology within Chapter 5.

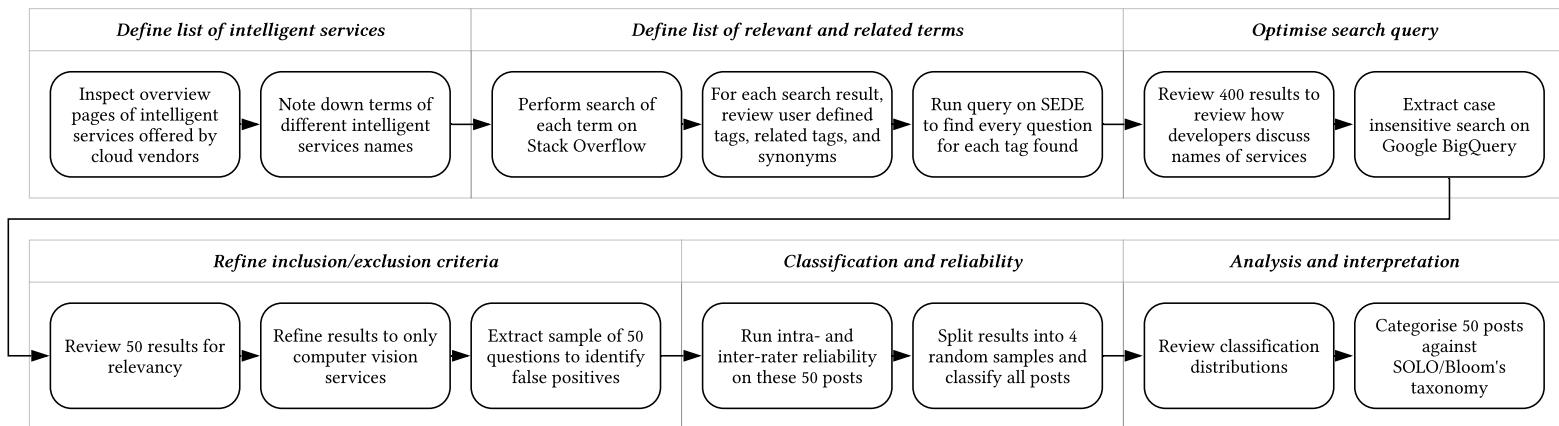


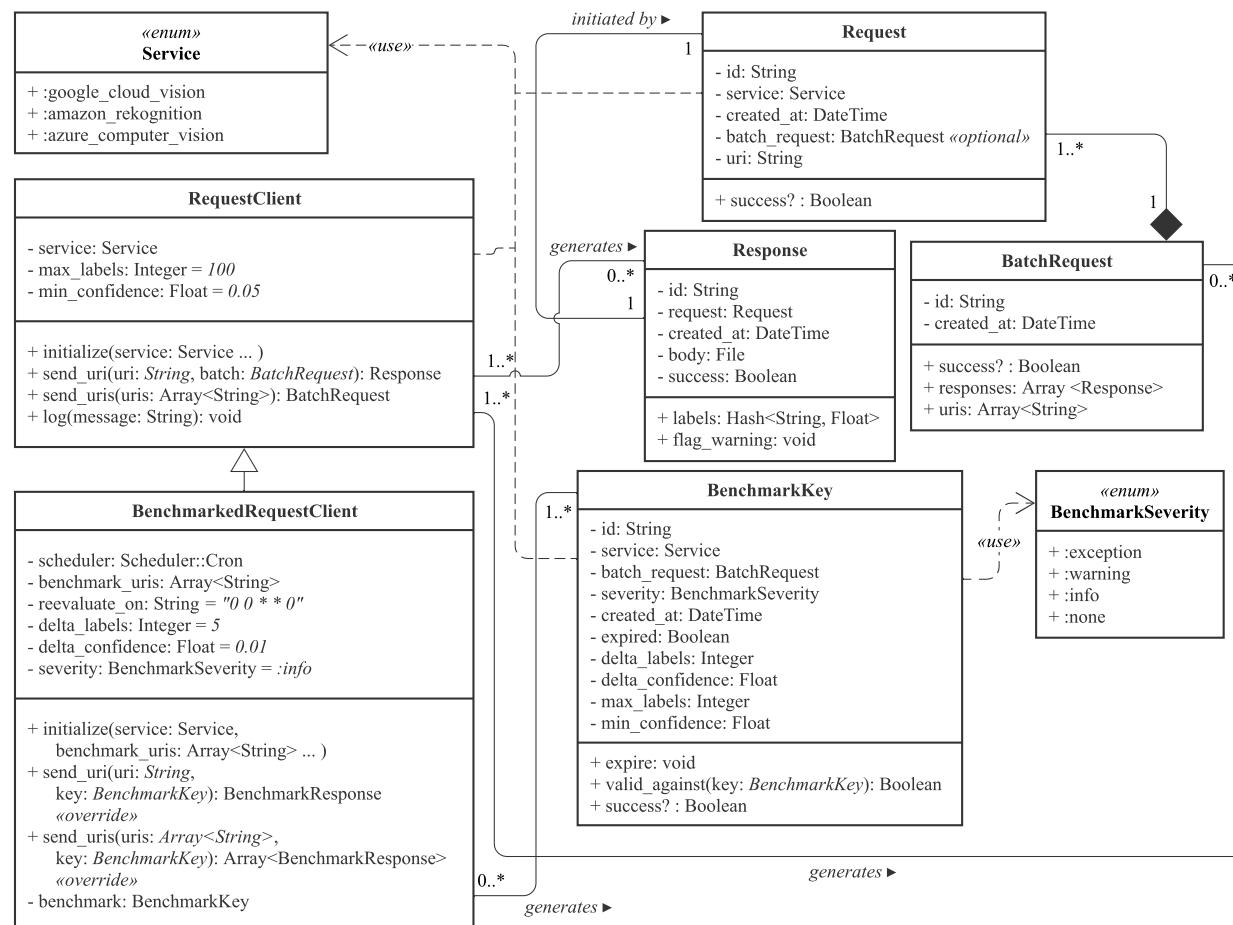
Figure A.11: Class diagram of the implementation of our architecture.

Figure A.12: Creation of a new benchmark proxy server using the architecture tactic.

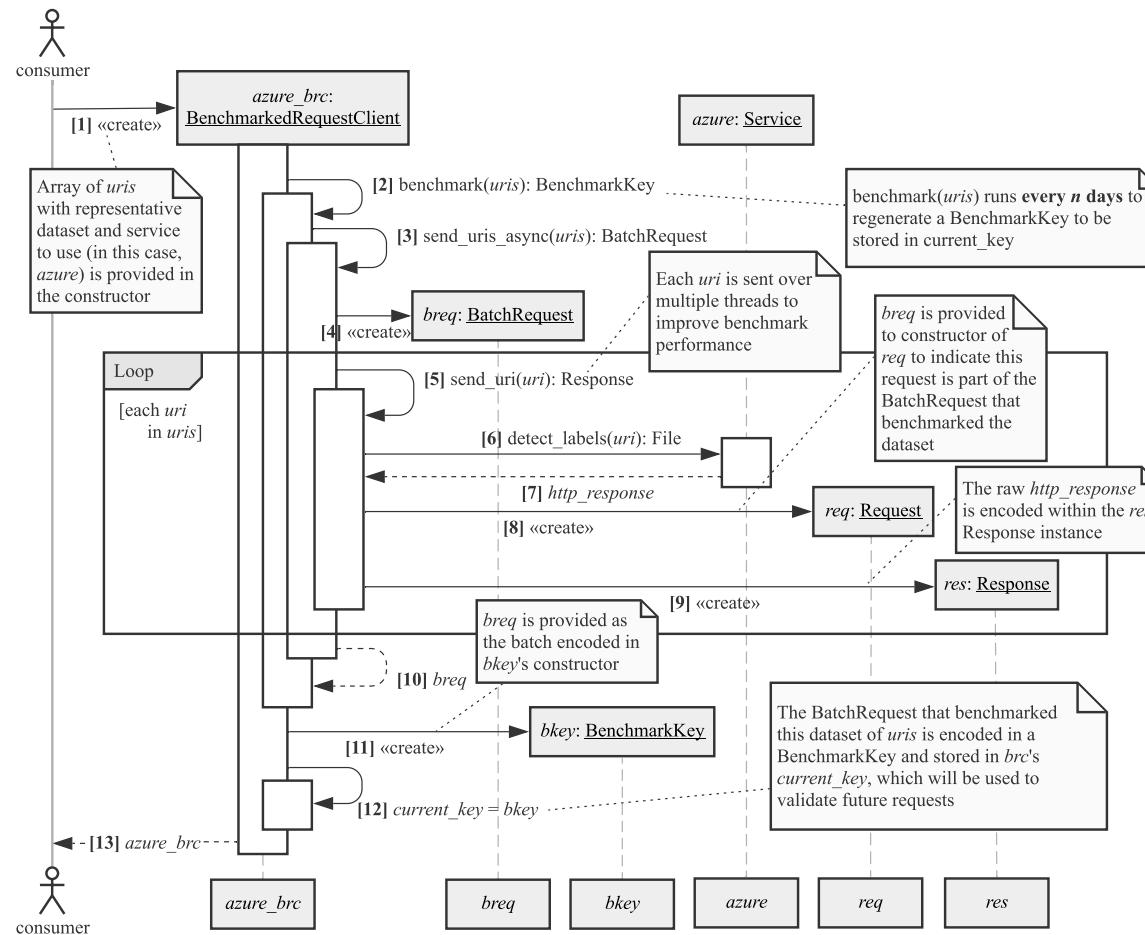


Figure A.13: Making a request through the proxy server ‘facade’.

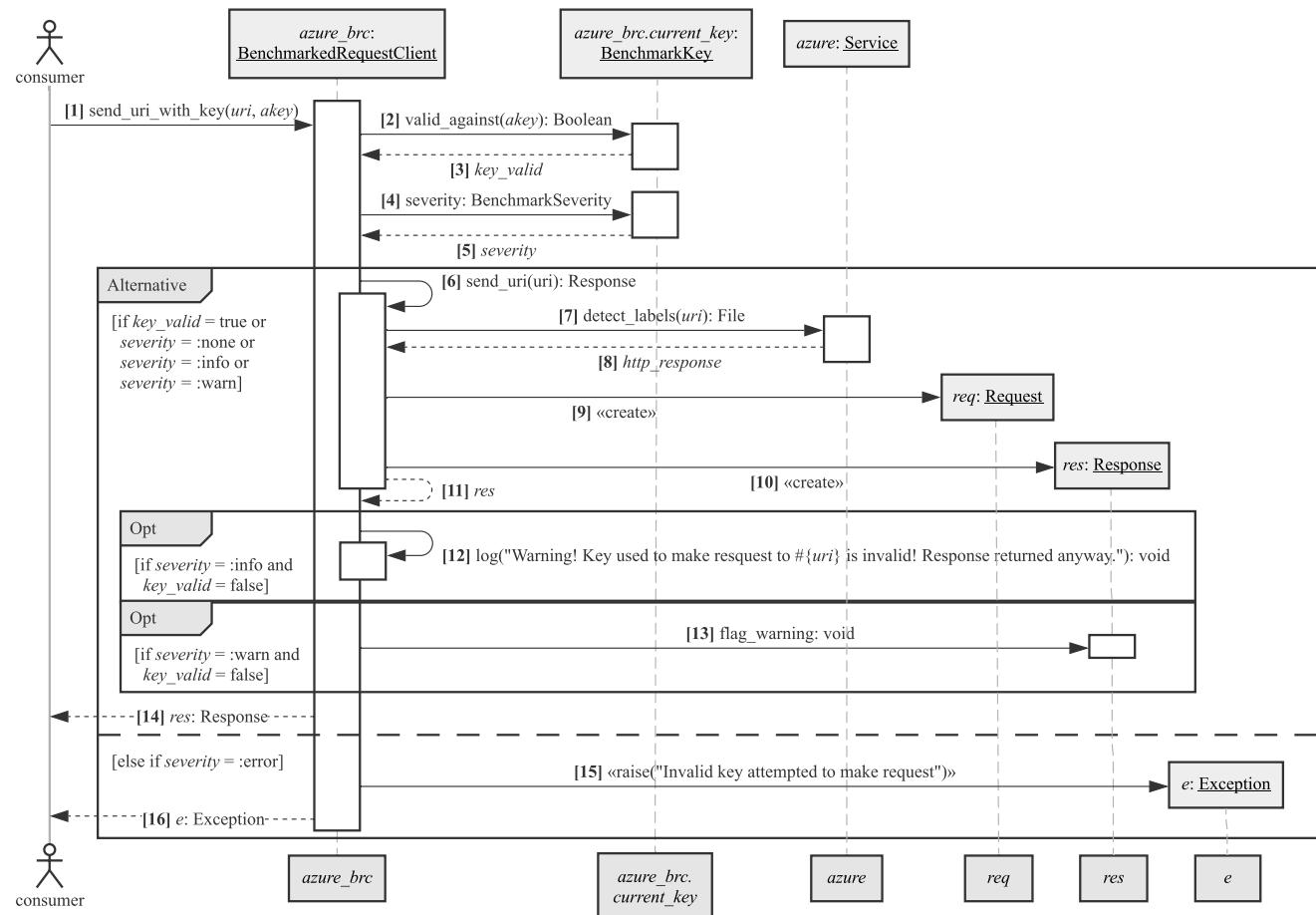


Figure A.14: State diagram of high-level workflows in the architectural tactic.

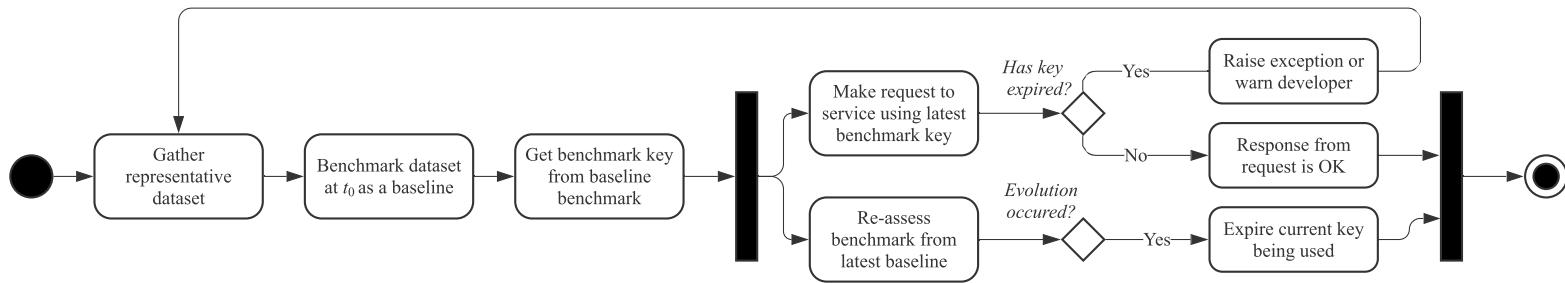


Figure A.15: Evolution occurring in the benchmark and how the architectural tactic notifies the consumer.

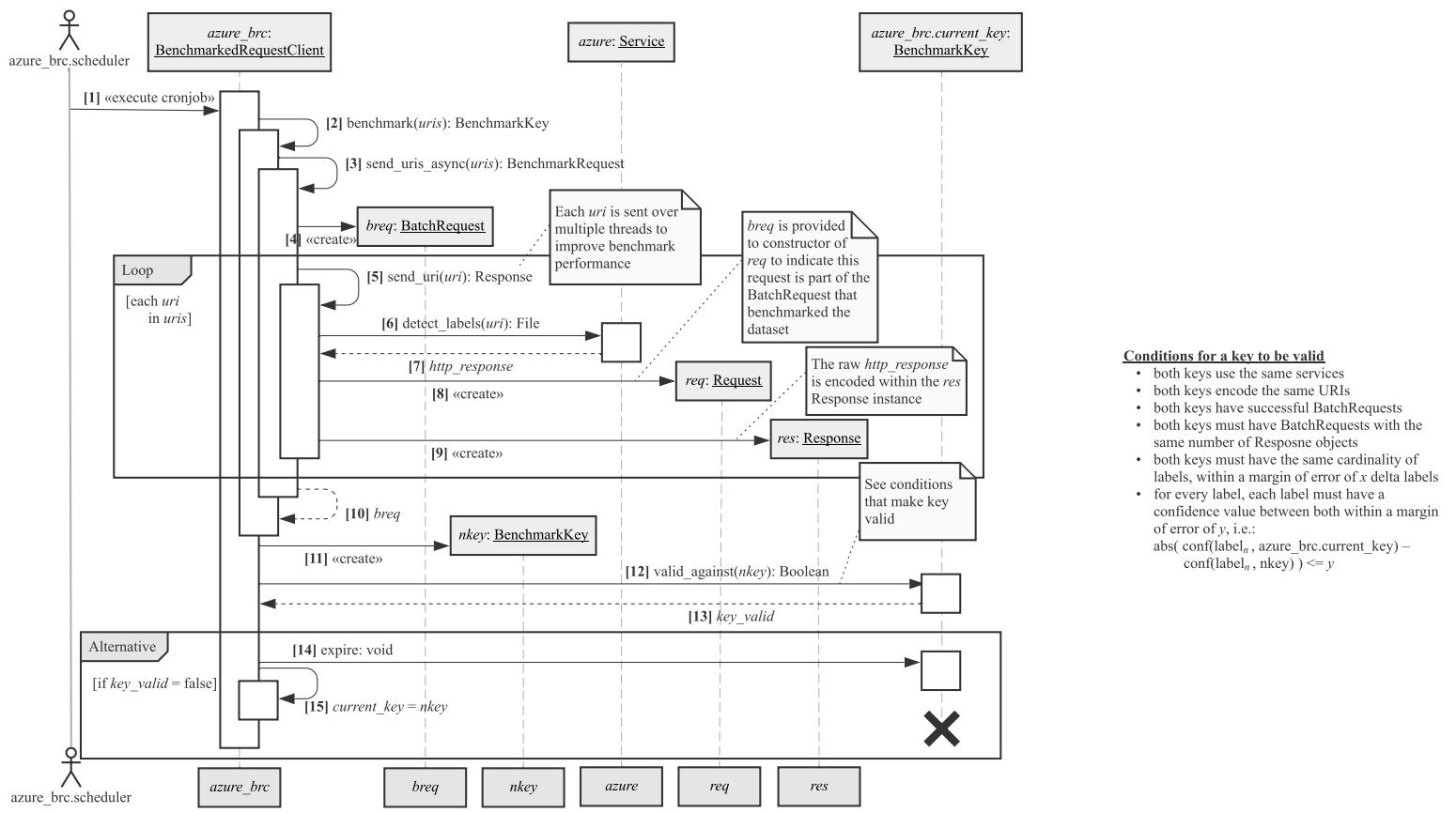
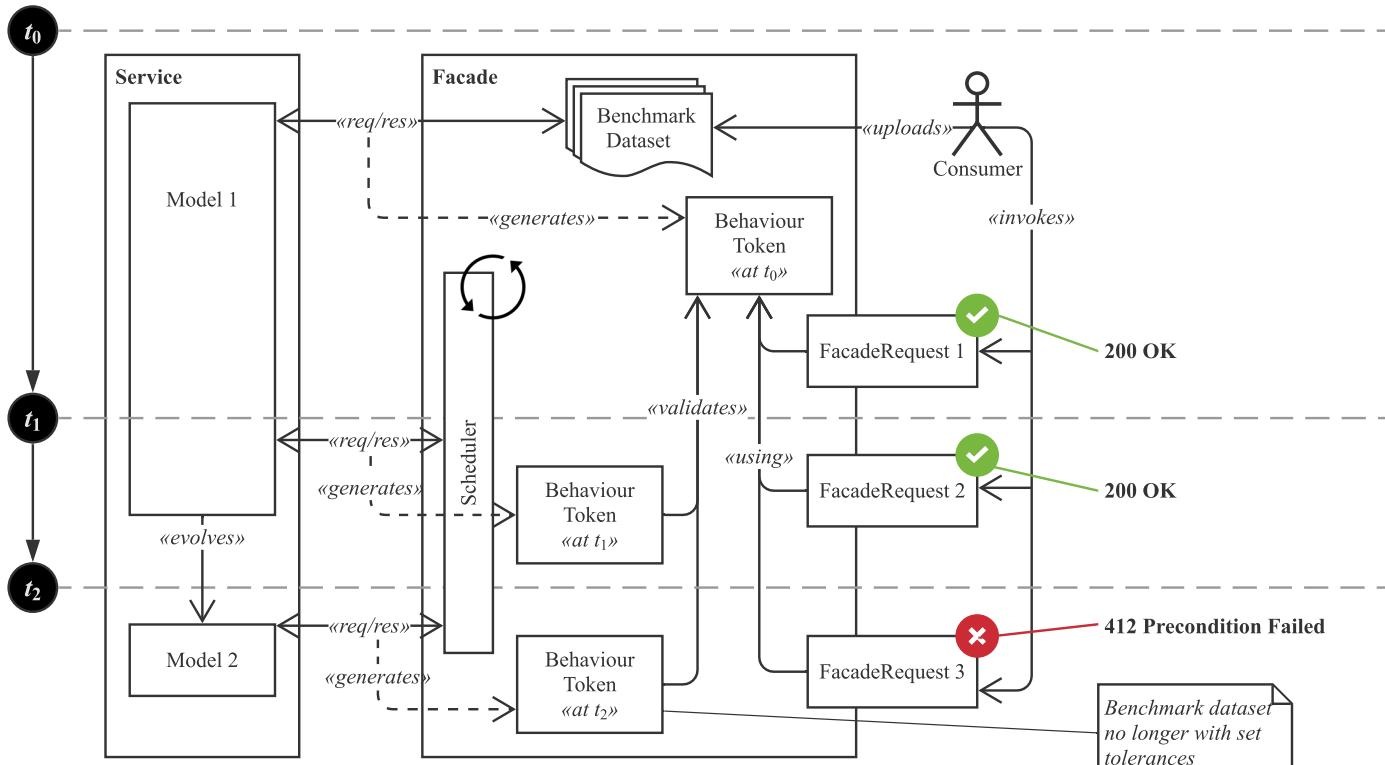


Figure A.16: Evolution occurring in an intelligent service and how the architectural tactic handles it.



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 
```

```

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 \"\
502               \"validation key (id=#{key.id}) service=#{key.service.name}."
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 \
528               \"different \"\
529               \"benchmark dataset URIS. The symmetric difference is: #{symm_diff_uris.\
530               \"to_a}\"."
531           }
532         )
533       end

```

```

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       response
886     end
887
888     # Sends a batch request with multiple images to client's respective service
889     # endpoint. Does *not* validate a response against a key (see
890     # ICVSB::BenchmarkedRequestClient).
891     # @param [Array<String>] uris An array of URIs to an image to detect labels.
892     # @return [BatchRequest] The batch request that was created.
893     def send_uris(uris)
894       raise ArgumentError, 'URIs must be an array of strings.' unless uris.is_a?(
895         ↪ Array)
896   
```

```

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         key.id}")

```

```

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
1347   # If benchmark trigger on num failures is set
1348   if @benchmark_config[:trigger_on_failcount].positive? &&
1349     @invalid_state_count > @benchmark_config[:trigger_on_failcount]
1350     ICVSB.linfo("Benchmark has failed #{@benchmark_config[:.
1351       ↪ trigger_on_failcount]} \"\
1352         'times... retriggering benchmark...'")
1353     @invalid_state_count = 0
1354     trigger_benchmark
1355   end
1356
1357   # Response behaviour is dependent on the severity encoded within the key
1358   case @current_key.benchmark_severity
1359   when BenchmarkSeverity::EXCEPTION
1360     # Only expose errors if they exist
1361     if (result[:key_errors].nil? || result[:key_errors].empty?) &&
1362       result[:response_errors].nil? &&
1363       result[:service_error].nil?
1364       result
1365     else
1366       {
1367         key_errors: result[:key_errors],
1368         response_errors: result[:response_errors],
1369         service_error: result[:service_error]
1370       }
1371     end
1372   when BenchmarkSeverity::WARNING
1373     # Flag a warning to the warning endpoint about this result if sev is WARN
1374     _flag_warning(result)
1375   end
1376 end

```

```

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))
461 | end
```

APPENDIX C

Supplementary Materials to Chapter 7

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-javastable 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 15737551
- [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 7.5.1. 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

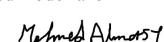
Author 4

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

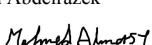
Author 3

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

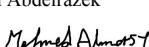
Author 4

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

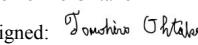
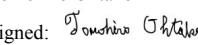
I agree to be named as one of the authors of this work, and confirm:

- 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:

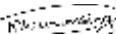
- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these 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