

Improved Usage of Pre-Trained Machine Learning Models Abstracted through Software Components

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

January 12, 2021

ABSTRACT

Software components hide implementation complexity by exposing a designed interface that permits easy integration and use. The explosive demand and interest in artificial intelligence (AI) and deep learning has led to creation of software components that offer various machine learning (ML) functions. The promise is that these AI components improve productivity and application developers can use them without a deep understanding of their underlying mechanics. Application developers currently have access to multiple AI components with a prominent focus on visual object recognition, natural language processing, audio analysis, anomaly detection and forecasting from numerical data. Simplified variations of these components are offered via cloud computing as intelligent web services; these services are often marketed as ‘developer friendly’ ML with the claim of being just another component accessible on the cloud through a web-based RESTful API.

A developer’s conceptual understanding of components they use impacts the internal and external quality of software they produce. Hence, vendors of intelligent web services must give sufficient level of conceptual detail to enable integration and effective use of their pre-packaged capabilities, ultimately to help developers who integrate with their services produce high-quality software.

This thesis investigates these emerging intelligent web services. Based on an analysis of the observable behaviour of intelligent web services, we show that their probabilistic results and evolution is not effectively communicated in the documentation. Our work shows that developers interpret and use these services using anchors built upon their understanding of traditional (i.e., deterministic and non-probabilistic) software components. We show how this mismatch results in a weak conceptual understanding of highly-abstracted forms of ML, impacting software quality. To mitigate documentation issues, we propose a taxonomy of the key requirements of good API documentation, which we derive from existing literature and triangulated through a survey with developers. We use this information to assess the value placed by developers on each API documentation artefact and identify gaps in the services’ documentation, which can be improved to assist conceptual understanding. Additionally, we propose an architectural tactic designed to reduce and guard against common issues identified when ML becomes highly-abstracted. The proposed tactic is intended to better integrate conventional software components with probabilistic and non-deterministic intelligent web services, ultimately to improve overall solution robustness and, thus, software quality.

This thesis makes a substantial contribution to the software engineering discipline by showing the non-trivial implications to software quality resulting from improper usage of such services and offers a pathway to safer use of the exciting new advances from the field of AI and deep learning.

Candidate Declaration

I certify that the thesis entitled “*Improved Usage of Pre-Trained Machine Learning Models Abstracted through Software Components*” 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
January 12, 2021

Access to Thesis

I am the author of the thesis entitled “*Improved Usage of Pre-Trained Machine Learning Models Abstracted through Software Components*” 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
January 12, 2021

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 Bonacci, 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 many contributions 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, Reuben Wilson, Mahdi Babaei, Rodney Pilgrim, and Simon Vajda and 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
January 12, 2021

Statistics about this PhD

This PhD journey consisted of the following:

- 82,188 words;
- 38,281 lines of L^AT_EX;
- 7 accepted publications;
- 3 rejected publications;
- 4 conferences, 2 attended virtually;
- 827 days of candidature;
- 3.7 months of examination;
- 1 global pandemic; and,
- 21 years of since my first day of primary school.

“...Now what?”

— BLOAT, *FINDING NEMO* (2003)

Contents

Abstract	iii
Candidate Declaration	v
Access to Thesis	vii
Acknowledgements	xi
Statistics about this PhD	xiii
Contents	xv
List of Publications	xxi
List of Abbreviations	xxiii
List of Figures	xxvii
List of Tables	xxxi
List of Listings	xxxiv
I Preface	1
1 Introduction	3
1.1 Research Context	7
1.2 Motivating Scenarios	9
1.2.1 Low Risk Motivating Scenario	10
1.2.2 High Risk Motivating Scenario	12
1.3 Research Motivation	14
1.3.1 Outputs are Probabilities	14

1.3.2	Evolution of Datasets	15
1.3.3	Selecting Appropriate Decision Boundaries	15
1.3.4	Documentation of the above concerns	16
1.4	Research Goals	16
1.5	Research Methodology	18
1.6	Thesis Organisation	19
1.6.1	Part I: Preface	19
1.6.2	Part II: Publications	19
1.6.3	Part III: Postface	23
1.6.4	Part IV: Appendices	23
1.7	Research Contributions	23
1.7.1	Contribution 1: Landscape Analysis & Preliminary Solutions	25
1.7.2	Contribution 2: Improving Documentation Attributes	25
1.7.3	Contribution 3: Service Integration Architecture	26
2	Background	29
2.1	Software Quality	30
2.1.1	Validation and Verification	31
2.1.2	Quality Attributes and Models	33
2.1.3	Reliability in Computer Vision	35
2.2	Probabilistic and Nondeterministic Systems	36
2.2.1	Interpreting the Uninterpretable	36
2.2.2	Explanation and Communication	39
2.2.3	Mechanics of Model Interpretation	40
2.3	Application Programming Interfaces	41
2.3.1	API Usability	41
2.4	Summary	42
3	Research Methodology	45
3.1	Research Questions Revisited	45
3.1.1	Empirical Research Questions	46
3.1.2	Non-Empirical Research Questions	47
3.2	Philosophical Stances	47
3.3	Research Methods	49
3.3.1	Review of Relevant Research Methods	49
3.3.2	Review of Data Collection Techniques for Field Studies	51
3.4	Research Design	51
3.4.1	Landscape Analysis of Computer Vision Services	51
3.4.2	Utility of API Documentation in Computer Vision Services	53
3.4.3	Developer Issues concerning Computer Vision Services	53
3.4.4	Designing Improved Integration Strategies	54

II Publications	55
4 Identifying Evolution in Computer Vision Services	57
4.1 Introduction	57
4.2 Motivating Example	59
4.3 Related Work	60
4.3.1 External Quality	60
4.3.2 Internal Quality	61
4.4 Method	63
4.5 Findings	65
4.5.1 Consistency of top labels	65
4.5.2 Consistency of confidence	68
4.5.3 Evolution risk	68
4.6 Recommendations	70
4.6.1 Recommendations for IWS users	70
4.6.2 Recommendations for IWS providers	71
4.7 Threats to Validity	73
4.7.1 Internal Validity	73
4.7.2 External Validity	73
4.7.3 Construct Validity	74
4.8 Conclusions & Future Work	74
5 Interpreting Pain-Points in Computer Vision Services	77
5.1 Introduction	77
5.2 Motivation	79
5.3 Background	81
5.4 Method	82
5.4.1 Data Extraction	82
5.4.2 Data Filtering	84
5.4.3 Data Analysis	85
5.5 Findings	87
5.5.1 Post classification and reliability analysis	87
5.5.2 Developer Frustrations	88
5.5.3 Statistical Distribution Analysis	90
5.6 Discussion	90
5.6.1 Answers to Research Questions	90
5.6.2 The Developer’s Learning Approach	92
5.6.3 Implications	95
5.7 Threats to Validity	97
5.7.1 Internal Validity	97
5.7.2 External Validity	97
5.7.3 Construct Validity	98
5.8 Conclusions	98

6 Ranking Computer Vision Service Issues using Emotion	99
6.1 Introduction	99
6.2 Motivation	101
6.3 Methodology	102
6.3.1 Dataset	102
6.3.2 Additional Dataset Cleansing	103
6.3.3 Automatic Emotion Classification	103
6.3.4 Manual Emotion Classification	104
6.3.5 Comparing Manual and Automatic Classification Methods	104
6.4 Findings	104
6.5 Discussion	106
6.6 Threats to Validity	108
6.6.1 Internal validity	108
6.6.2 External validity	108
6.6.3 Construct validity	109
6.7 Conclusion	109
7 Using Emotion Classification Models against Stack Overflow	111
7.1 Introduction	111
7.2 Motivation	113
7.3 Method	113
7.4 Results	115
7.4.1 Limitations of the Text Classifier	115
7.4.2 Data imbalance	115
7.4.3 Emotion Labeling Bias	118
7.4.4 Emotion Labelling and Classification Granularity	118
7.5 Discussion	119
7.6 Threats to Validity	120
7.7 Related Work	121
7.8 Conclusion	121
8 Better Documenting Computer Vision Services	123
8.1 Introduction	123
8.2 Related Work	126
8.2.1 Systematic Reviews in Software Documentation	126
8.2.2 API Usability and Documentation Knowledge	127
8.2.3 Computer Vision Services	129
8.3 Taxonomy Development	129
8.3.1 Systematic Mapping Study	129
8.3.2 Development of the Taxonomy	134
8.4 A Taxonomy for API Documentation	137
8.5 Validating the Taxonomy	139
8.5.1 Survey Study	139
8.5.2 Empirical Application on Computer Vision Services	142
8.6 Taxonomy Analysis	142

8.6.1	Exploring IPS and ILS Values	143
8.6.2	Triangulating IPS, ILS and Computer Vision	146
8.6.3	Recommendations Resulting from Analysis	148
8.7	Threats to Validity	152
8.7.1	Internal Validity	152
8.7.2	External Validity	152
8.7.3	Construct Validity	153
8.8	Conclusions & Future Work	154
9	Using a Facade Pattern to combine Computer Vision Services	157
9.1	Introduction	157
9.1.1	Motivating Scenario: Intelligent vs Traditional Web Services	158
9.1.2	Research Motivation	159
9.2	Merging API Responses	159
9.2.1	API Facade Pattern	160
9.2.2	Merge Operations	160
9.2.3	Merging Operators for Labels	161
9.3	Graph of Labels	162
9.3.1	Labels and synsets	162
9.3.2	Connected Components	162
9.4	API Results Merging Algorithm	165
9.4.1	Mapping Labels to Synsets	165
9.4.2	Deciding Total Number of Labels	165
9.4.3	Allocating Number of Labels to Connected Components . .	166
9.4.4	Selecting Labels from Connected Components	167
9.4.5	Conformance to properties	167
9.5	Evaluation	167
9.5.1	Evaluation Method	167
9.5.2	Naive Operators	168
9.5.3	Traditional Proportional Representation Operators . . .	170
9.5.4	New Proposed Label Merge Technique	170
9.5.5	Performance	170
9.6	Conclusions and Future Work	171
10	Supporting Safe Usage of Intelligent Web Services	173
10.1	Introduction	173
10.2	Motivating Example	176
10.3	Threshy	178
10.4	Related work	179
10.5	Conclusions & Future Work	180
11	An Integration Architecture Tactic to Guard AI-first Components	183
11.1	Introduction	183
11.2	Motivating Example	186
11.3	Intelligent Services	187
11.3.1	‘Intelligent’ vs ‘Traditional’ Web Services	187

11.3.2 Dimensions of Evolution	187
11.3.3 Limited Configurability	188
11.4 Our Approach	191
11.4.1 Core Components	191
11.4.2 Usage Example	196
11.5 Evaluation	197
11.5.1 Data Collection and Preparation	198
11.5.2 Results	198
11.5.3 Threats to Validity	200
11.6 Discussion	204
11.6.1 Implications	204
11.6.2 Limitations	204
11.6.3 Future Work	205
11.7 Related Work	206
11.8 Conclusions	207
 III Postface	 209
 12 Conclusions & Future Work	 211
12.1 Contributions of this Work	211
12.1.1 Answers to Research Questions	212
12.1.2 Limitations to Research Answers & Future Research	216
12.2 Concluding Remarks	218
 References	 241
 List of Online Artefacts	 243
 IV Appendices	 247
 A Additional Materials	 249
A.1 Development, Documentation and Usage of Web APIs	251
A.2 Additional Figures	255
 B Reference Architecture Source Code	 269
 C Supplementary Materials to Chapter 8	 303
C.1 Detailed Overview of Our Proposed Taxonomy	305
C.2 Sources of Documentation	309
C.3 List of Primary Sources	312
C.4 Detailed Suggested Improvements	316
C.4.1 Dimension A Issues	316
C.4.2 Dimension B Issues	318
C.4.3 Dimension C Issues	318
C.4.4 Dimension D Issues	319

C.5 Survey Questions	320
D Authorship Statements	325
E Ethics Clearance	365

List of Publications

Below lists publications arising from work completed in this PhD.

1. A. Cummaudo, R. Vasa, and J. Grundy, “Requirements of API Documentation: A Case Study into Computer Vision Services,” *IEEE Transactions on Software Engineering*, pp. 1–1, 2020, DOI 10.1109/TSE.2020.3047088
2. A. Cummaudo, S. Barnett, R. Vasa, J. Grundy, and M. Abdelrazek, “Beware the evolving ‘intelligent’ web service! An integration architecture tactic to guard AI-first components,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event, USA: ACM, November 2020. DOI 10.1145/3368089.3409688, pp. 269–280
3. A. Cummaudo, S. Barnett, R. Vasa, and J. Grundy, “Threshy: Supporting Safe Usage of Intelligent Web Services,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event, USA: ACM, November 2020. DOI 10.1145/3368089.3417919, pp. 1645–1649
4. 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
5. 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
6. 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
7. 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 10.1007/978-3-03-19274-7_28. ISBN 978-3-03-019273-0. ISSN 1611-3349 pp. 391–406

List of Abbreviations

A²I² Applied Artificial Intelligence Institute. 51, 53

AI artificial intelligence. iii, 3–7, 10, 14, 16, 36, 38, 39, 57, 58, 61, 62, 72, 75, 77–79, 81, 82, 92, 95, 96, 98, 99, 111–113, 118–121, 123, 124, 154, 155, 159, 160, 183, 184, 187, 200, 207, 214, 216, 218

API application programming interface. iii, 3–18, 20, 22, 25, 26, 29, 30, 32, 33, 41–43, 46–48, 50, 53, 58, 59, 62, 64, 71–74, 77–84, 87–100, 102–105, 107, 108, 111, 113, 119, 123–129, 131–155, 157–160, 162, 164, 165, 167, 168, 171, 174, 180, 183, 185, 187, 190–192, 198, 204, 207, 211, 213–215, 251, 254

BYOML Build Your Own Machine Learning. 7, 8

CC connected component. 162, 165–167, 171

CDSS clinical decision support system. 10, 12, 13

CNN convolutional neural network. 12, 13, 36, 60

CRUD create, read, update, and delete. 254

CVS computer vision service. 4, 9–12, 14, 16–20, 22–27, 29, 31, 32, 35, 41–43, 46–48, 50, 51, 53, 57–63, 66, 71, 73, 74, 77, 79, 84, 89, 93, 97–99, 101, 102, 104, 108, 109, 112, 113, 118, 123–127, 129, 136, 137, 139, 140, 142, 146–149, 151–155, 157, 159, 160, 162, 165, 167, 171, 174, 184, 185, 188, 190, 191, 198, 200, 205, 207, 211–216, 255

DCE distributed computing environment. 251

DSM Distributional Semantic Model. 113

HITL human-in-the-loop. 13

- HTML** Hypertext Markup Language. 84
- HTTP** Hypertext Transfer Protocol. 8, 195, 196, 199, 204, 251, 253, 254
- IDE** integrated development environment. 34
- IDL** interface definition language. 251, 254
- ILS** In-Literature Score. 125, 137, 142–146, 151, 153, 154
- IPS** In-Practice Score. 125, 137, 141–147, 154
- IRR** inter-rater reliability. 97
- IWS** intelligent web service. 6, 7, 9, 11, 13, 14, 16, 17, 19, 20, 22, 23, 29–33, 35, 36, 41–43, 57–59, 61, 62, 70, 72, 74, 75, 77–83, 85, 87, 88, 90–100, 108, 111, 112, 121, 122, 155, 157–159, 171, 173, 174, 176, 177, 179, 180, 183, 184, 186, 187, 190–193, 204–207, 211, 216–218, 255
- JSON** JavaScript Object Notation. 9, 174, 188, 196, 198
- ML** machine learning. iii, 3–8, 10, 11, 14, 16, 17, 20, 22, 25, 33, 38, 41, 43, 57, 58, 61, 62, 72, 74, 78–80, 82, 83, 95, 96, 99, 103, 108, 109, 117, 119, 157, 158, 171, 173, 174, 178, 180
- NN** neural network. 15, 36–38, 40
- PaaS** Platform as a Service. 9, 13, 61
- QoS** quality of service. 61, 62, 251
- RAML** RESTful API Modeling Language. 254
- REST** REpresentational State Transfer. iii, 7, 58, 77, 98, 123, 158, 183, 207, 252, 254
- ROI** region of interest. 12, 13
- RPC** remote procedure call. 251
- SDK** software development kit. 59, 89, 131, 148
- SLA** service-level agreement. 61, 251
- SMS** systematic mapping study. 22, 23, 26, 125–129, 135, 144, 145, 150, 152, 154, 155
- SO** Stack Overflow. 7, 17, 20, 25, 26, 46, 47, 50, 53, 54, 62, 63, 77, 79–83, 85, 87, 88, 90–93, 95–104, 106, 108, 109, 111–115, 117, 118, 120–122, 214

- SOA** service-oriented architecture. 251
- SOAP** Simple Object Access Protocol. 7, 251–254
- SOLO** Structure of the Observed Learning Outcome. 92–94, 96, 97
- SQA** service quality assurance. 59, 60
- SQuaRE** Systems and software Quality Requirements and Evaluation. 34
- SUS** System Usability Scale. 18, 125, 139, 141
- SVM** support vector machine. 36, 40, 113
- SWEBOK** Software Engineering Body of Knowledge. 131, 132, 135
- URI** uniform resource identifier. 254
- V&V** verification & validation. 29–33, 42
- WADL** Web Application Description Language. 254
- WSDL** Web Services Description Language. 251
- XML** eXtendable markup language. 9, 251

List of Figures

1.1	Increasing interest in the developer community of computer vision services	4
1.2	Differences between data- and procedural-driven cloud services	6
1.3	The spectrum of machine learning	8
1.4	Overview of intelligent web services	9
1.5	CancerAssist Context Diagram	13
1.6	Overview publication coherency	24
2.1	Mindset clashes within the development, use and nature of a IWS	30
2.2	Leakage of internal and external quality in	33
2.3	Overview of software quality models	34
2.4	Adversarial examples in computer vision	37
2.5	Deterministic versus nondeterministic systems	38
2.6	Theory of AI communication	40
4.1	Consistency of labels in computer vision services is rare	65
4.2	Top labels for images between computer vision services do not intersect .	66
4.3	Computer vision services can return multiple top labels	67
4.4	Cumulative distribution of top label confidences	69
4.5	Cumulative distribution of intersecting top label confidences	69
4.6	Agreement of labels between multiple computer vision services do not share similar confidences	70
5.1	Traits of intelligent web services compared to DIY ML	80
5.2	Trend of Stack Overflow posts discussing computer vision services .	81
5.3	Comparing documentation-specific and generalised classifications of Stack Overflow posts	88
5.4	Alignment of Bloom and SOLO taxonomies against computer vision issues	94
6.1	Distributions of the types of questions raised	104

6.2 Proportions of emotions per question type	106
7.1 Emotion classifier training data imbalance	117
8.1 Systematic mapping study search results, by years	130
8.2 Filtering steps used in the systematic mapping study	130
8.3 A systematic map of API documentation studies	135
8.4 Our proposed API documentation taxonomy	138
8.5 Roles and seniority from survey participants	140
8.6 Comparing value of API documentation artefacts to developers vs research attention	145
8.7 Comparing value of API documentation artefacts to presence in Computer Vision Services	147
9.1 Overview of the proposed facade	160
9.2 Graph of associated synsets against two different endpoints	163
9.3 Label counts per API assessed	164
9.4 Connected components vs. images	164
9.5 Allocation to connected components	166
9.6 F-measure comparison	170
10.1 Request and response for computer vision services provide limited configurability	174
10.2 Example case study of evaluating model performance in two different models	175
10.3 Threshy supports threshold selection and monitoring	176
10.4 Example pipeline of a computer vision system	177
10.5 UI workflow of Threshy	178
10.6 Architecture of Threshy	180
11.1 Prominent computer vision service evolution	185
11.2 Dimensions of evolution within computer vision services	188
11.3 Example of substantial confidence change	188
11.4 Directly versus indirectly accessing intelligent services	189
11.5 Sample request and response for intelligent services	190
11.6 State diagram of architecture workflows	194
11.7 Precondition failure taxonomy	195
11.8 Histogram of confidence variation	199
11.9 Architecture response to substantial confidence evolution	201
11.10 Architecture response to label set evolution	202
11.11 Architecture response of expected label mismatch	203
12.1 Results from computer vision services can be disparate and non-static	213
12.2 Distribution of issues on Stack Overflow	215
A.1 SOAP versus REST search interest over time	252

A.2	Causal factors that may influence understanding of intelligent web services	257
A.3	A proposal technical domain model for intelligent services	258
A.4	Potential questions that can be asked around causal factors of a developer's understanding of an intelligent service	259
A.5	Threshy and developer interaction with decision boundaries	259
A.6	Threshy domain model	260
A.7	Threshy sequence diagram	260
A.8	High-level overview of our method in Chapter 5	261
A.9	Class diagram of architecture implementation	262
A.10	Creation of a benchmark using the architecture tactic	263
A.11	Making a request via the proxy server facade	264
A.12	High-level workflow of the architectural tactic	265
A.13	Handling of evolution using our architecture (i)	266
A.14	Handling of evolution using our architecture (ii)	267

List of Tables

1.1	Categorisation of AI-based products and services	5
1.2	Differing characteristics of cloud services	6
1.3	Comparison of the machine learning spectrum	8
1.4	Varying confidence changes over time between three computer vision services	11
1.5	Definitions of ‘confidence’ in CV documentation	12
1.6	List of publications resulting from this thesis	21
3.1	Classification of research questions in this thesis	46
3.2	Review of field study techniques	52
4.1	Characteristics of data in computer vision evolution assessment . . .	64
4.2	Ratio of consistent labels in computer vision services	65
4.3	Evolution of top labels and confidence values	68
5.1	Taxonomies used in our Stack Overflow mining study	86
5.2	Example Stack Overflow posts aligning to Bloom’s and SOLO taxonomies	93
6.1	Our interpretations from a Stack Overflow question type taxonomy .	102
6.2	Frequency of emotions per question type.	105
6.3	Inter-rater agreement between human and automatic classification .	106
6.4	Sample Stack Overflow questions with emotions identified	107
7.1	Emotion classification frequencies	114
7.2	Reliability Analysis of Emotion Classification	114
7.3	Sample questions comparing automated and human rater classifications	116
8.1	Summary of search results in API documentation	132
8.2	Data extraction form used for the systematic mapping study	134
8.3	Intervals assigned to ILS and IPS values	143

8.4 Documentation artefacts of high value to developers that are under-researched and under-documented	149
9.1 Statistics for the number of labels	162
9.2 First allocation iteration	167
9.3 Second allocation iteration	167
9.4 Third allocation iteration	167
9.5 Fourth allocation iteration	167
9.6 Matching to human-verified labels	169
9.7 Evaluation results of the facade	169
9.8 Average of evaluation result of the facade	169
11.1 Potential reasons for precondition failure	191
11.2 Rules encoded within behaviour tokens	193
11.3 Variance in ontologies	200

List of Listings

A.1	An example SOAP request	253
A.2	An example SOAP response	253
A.3	An example RESTful request	254
A.4	An example RESTful response	254
B.1	Implementation of the architecture module components	269
B.2	Implementation of the architecture facade API	295

Part I

Preface

CHAPTER 1

3

4

5

Introduction

6

7 Abstraction layers are the application developer’s productivity powerhouse as de-
8 vellers need not continuously consider underlying mechanics. The ubiquitous ap-
9 plication programming interface (API) enables separation of concerns and reusable
10 component interaction; for example, complex graphics rendering and image manip-
11 ulation is all achievable via a half-dozen lines of code with appropriate libraries and
12 frameworks, for example OpenCV’s API.

13 ML, too, is being abstracted and offered behind APIs. The 2010s have shown
14 an explosion of cloud-based services providing *web* APIs typically marketed under
15 an AI banner. The ML algorithms, data processing pipelines, and infrastructure
16 bringing these techniques to life are also abstracted behind APIs calls, driven by
17 the motivation to make it easier for developers to blend AI into their software.
18 There is an explosion of interest from application developers (see Figure 1.1) that
19 are investigating and exploring how best to infuse recent advances in AI into their
20 software systems. Combined with an ever-increasing buffet of AI-based solutions,
21 technologies and products (see Table 1.1) for developers to choose from, it is evident
22 that we are at the cusp of a new generation of ‘AI-first’ software.

23 Application developers build procedural and functional applications, where code
24 typically evaluates deterministically to produce outcomes. Such software does not
25 rely on probabilistic behaviour, unlike AI-first software where, often, ML techniques
26 are employed. However, application developers, accustomed to such traditional
27 software engineering paradigms, may not be aware of potential side-effects of those
28 probabilistic techniques. Software that leverages recent advances in AI—more
29 specifically data-driven ML techniques—will often have a layer of rules that wrap
30 the ML components. AI-first software is, however, not *solely* procedural-driven
31 and combines large datasets with rules to produce outcomes. Therefore, they are
32 both *data-driven* and procedural-driven. The consequence is that large datasets—
33 that train ML models—combined with the algorithmic techniques behind these
34 models result in probabilistic behaviour. Further, since these models can continually

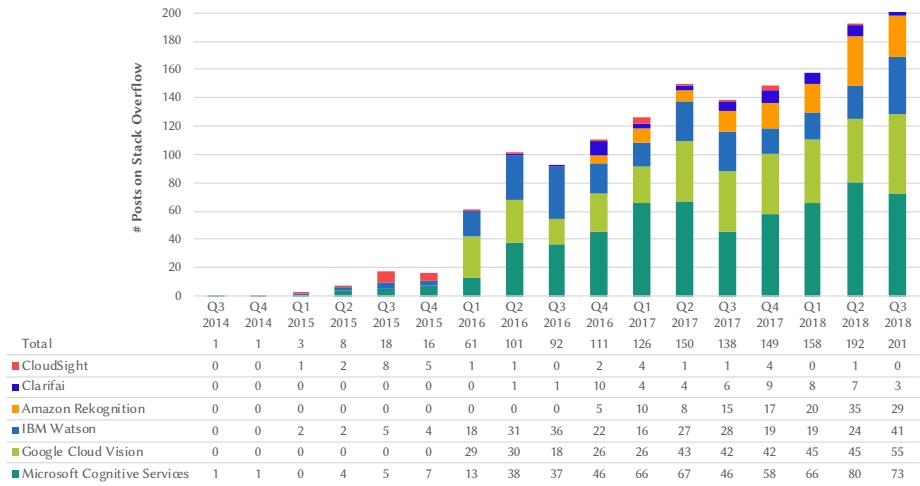


Figure 1.1: Increasing interest within the developer community for computer vision services (CVSs) is shown via Stack Overflow posts. These trends of CVS usage were measured as discussion of posts tagged with the relevant product name.¹ This graph is based on data from Chapter 5.

35 learn from *new* data with time, existing probabilistic behaviour can evolve and thus
36 regression testing techniques need to be adjusted as well for new data.

37 Developing AI-infused applications requires both code *and data*, and an applica-
38 tion developer can approach developing from three perspectives, further expanded
39 in Section 1.1:

- 40 1. The application developer defines an ML model from scratch and trains it from
41 a curated dataset. This approach is laborious in time and demands experience
42 and knowledge of ML methods, but the tradeoff is that they have full autonomy
43 in the models they create.
- 44 2. The application developer downloads a pre-trained model (e.g., YOLO [297]
45 for computer vision, or GPT-2 [293] for natural language processing) and
46 ‘plugs’ it into an existing ML framework, such as Tensorflow [1] or PyTorch
47 [275]. This approach removes the time taken to collect data, design and train
48 the ML model; the developers, still need to know where to find these models,
49 evaluate them, and then learn the frameworks² within which they operate to
50 use them effectively.
- 51 3. The application developer uses a cloud-based service. It is fast to integrate
52 into their applications, and the APIs offered abstract the technical know-how
53 behind a web call.

54 While much research has investigated these first two perspectives (see Chapter 2),
55 the third is yet to be deeply explored, despite the fact that vendors are promoting new
56 offerings encapsulated under this third perspective. As shown in Table 1.1, vendors

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.

Table 1.1: A broad range of AI-based vendors, products, and services is emerging in recent years. (Adapted from [222].)

Category	Sample Vendors & Products	Typical Use Cases
Embedded AI: Expert assistants leverage AI technology embedded in platforms and solutions.	Amazon: <i>Alexa</i> Apple: <i>Siri</i> Facebook: <i>Messenger</i> Google: <i>Google Assistant</i> Microsoft: <i>Cortana</i> Salesforce: <i>MetaMind</i>	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, and as bots or agents. Used in voice, image recognition, and various levels of natural language processing sophistication.
AI point solutions: Point solutions provide specialised capabilities for natural language processing, vision, speech, and reasoning.	24[7]: <i>24/7</i> Admantx: <i>Admantx</i> Affectiva: <i>Affdex</i> Assist: <i>AssistDigital</i> Automated Insights: <i>Wordsmith</i> Beyond Verbal: <i>Beyond Verbal</i> Expert System: <i>Cogito</i> HPE: <i>Haven OnDemand</i> IBM: <i>Watson Analytics</i> Narrative Science: <i>Quill</i> Nuance: <i>Dragon</i> Salesforce: <i>MetaMind</i> Wise.io: <i>Wise Support</i>	Semantic text, facial/visual recognition, voice intonation, intelligent narratives. Various levels of natural language processing, from brief text messaging, chat/conversational messaging, full complex text understanding. Machine learning, predictive analytics, text analytics/mining, knowledge management and search. Used as expert advisors, reasoning tools, or in customer service.
AI platforms: Platforms that offer various AI tech, including (deep) machine learning, as tools, APIs, or services to build solutions.	CognitiveScale: <i>Engage, Amplify</i> Digital Reasoning: <i>Synthesys</i> Google: <i>Google Cloud ML</i> IBM: <i>Watson Knowledge Studio</i> Intel: <i>Saffron Natural Intelligence</i> IPsoft: <i>Amelia, Apollo, IP Center</i> Microsoft: <i>Cortana Intelligence Suite</i> Nuance: <i>360 platform</i> Salesforce: <i>Einstein</i> Wipro: <i>Holmes</i>	APIs, cloud services, on-premises for developers to build AI solutions. Insights/advice building and rule-based reasoning. Vertical domain advisors (e.g., fraud detection in banking, financial advisors, healthcare). Cognitive services and bots.

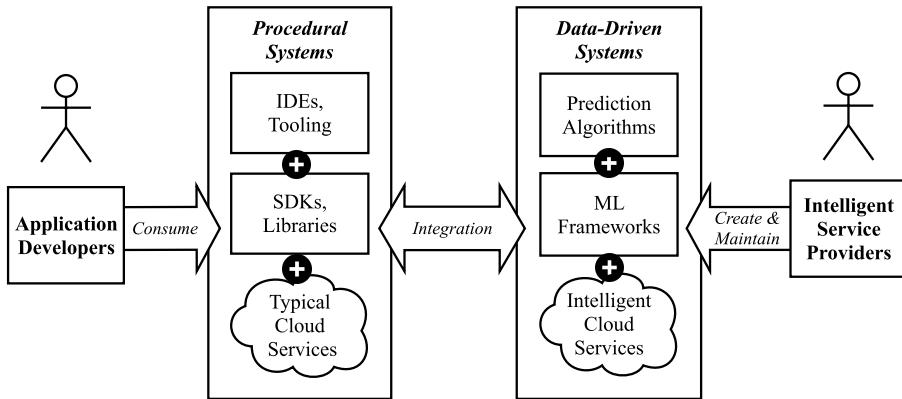


Figure 1.2: The application developer’s procedural-driven toolchain is distinct from data-driven toolchain. A developer must consume a typical, data-driven cloud service in a different way than an intelligent data-driven cloud service as they are not the same type of system.

⁵⁷ are rapidly pushing out new ML-based offerings in the form of cloud-based APIs
⁵⁸ end-points (AI platforms), where the API abstraction masks away the underlying me-
⁵⁹ chanics of the models. Developers that use these cloud-based services are presented
⁶⁰ with documentation providing a narrative (i.e., marketing and in the documentation)
⁶¹ that implies integration of these services are just like other cloud services. But does
⁶² this implication, coupled with abstractions that hide the assumptions made by the
⁶³ AI-service providers, lead to developer pain-points and miscomprehension? If so,
⁶⁴ how can the service providers improve their documentation to alleviate this? Do
⁶⁵ these data-driven services share similarities to the runtime behaviour of traditional
⁶⁶ cloud services? And if not, how best can the application developer integrate the
⁶⁷ data-driven service into their a procedural-driven application to produce AI-first
⁶⁸ software?

Table 1.2: Differing characteristics of intelligent and typical web services.

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

⁶⁹ Figure 1.2 provides an illustrative overview between the context clashing of
⁷⁰ procedural-driven applications and data-driven cloud services, and we contrast char-
⁷¹ acteristics of typical cloud systems and data-driven ones in Table 1.2.

 In this thesis, we show that (i) developers do not properly understand the probabilistic data-driven machine-learnt behaviour abstracted behind the end-points, (ii) the ‘intelligent behaviour’ is not fully contained and leaks into the applications that make use of these end-points, and finally (iii) we present how these concerns can be addressed via better documentation and software architecture. that the integration and developer comprehension of cloud services differ from the procedural-driven nature of end-applications.

1.1 Research Context

There are a range of integration techniques available to developers, as reflected by Google AI’s³ *machine learning spectrum* [207, 234, 268]. This range is grouped into the three tiers aforementioned, encompassing skills, effort, users, and types of outputs of integration techniques. At one extreme, this approach involves the academic research of developing algorithms and self-sourcing data to achieve intelligence—coined as Build Your Own Machine Learning (BYOML) [180, 234, 268]. The other extreme involves off-the-shelf, ‘friendlier’ (abstracted) intelligence with easy-to-use APIs targeted towards applications developers. The middle-ground involves a mix of the two, with varying levels of automation to assist in development, that turns custom datasets into machine intelligence. We illustrate the slightly varied characteristics within this spectrum in Table 1.3 and Figure 1.3.

These cloud AI-services are gaining traction within developer circles: we show an increasing trend of Stack Overflow posts mentioning intelligent computer vision services in Figure 1.1.⁴ Academia provides varied nomenclature for these services, such as *Cognitive Applications* and *Machine Learning Services* [367] or *Machine Learning as a Service* [300]. For the context of this thesis, we will refer to such services under broader term of **intelligent web services (IWSs)**,⁵ and diagrammatically express their usage within Figure 1.4.

There are many types of IWSs available to software developers, offering a range of functions, such as optical character recognition, text-to-speech and speech-to-text transcription, object categorisation, facial analysis and recognition, and natural language processing. The general workflow of using an IWS is more-or-less the same: a developer accesses an IWS component via REST/SOAP API(s), which is

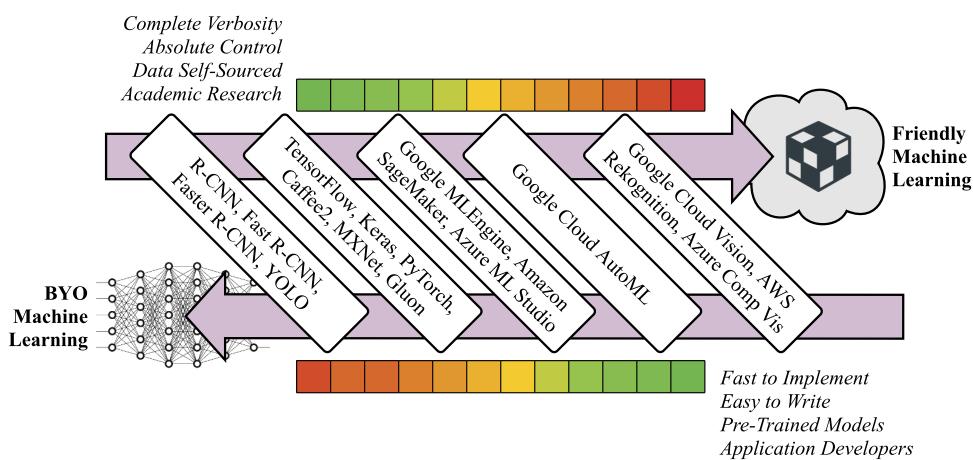
³Google AI was recently rebranded from Google Research, further highlighting how the ‘AI-first’ philosophy is increasingly becoming embedded in companies’ product lines and research and development teams. Spearheaded through work achieved at Google, Microsoft and Facebook, the emphasis on an AI-first attitude we see through Google’s 2018 rebranding of *Google Research* to *Google AI* [164] is evident. A further example includes how Facebook leverage AI *at scale* within their infrastructure and platforms [272].

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

⁵This term is an extension inspired by the term ‘web service’, as defined by the World Wide Web Consortium. See <https://bit.ly/2CQWJ2Z>, last accessed 19 July 2020.

Table 1.3: Comparison of the machine learning spectrum.

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

**Figure 1.3:** Examples within the ML spectrum of computer vision. Colour scales indicates the benefits (green) and drawbacks (red) of each end of the spectrum.

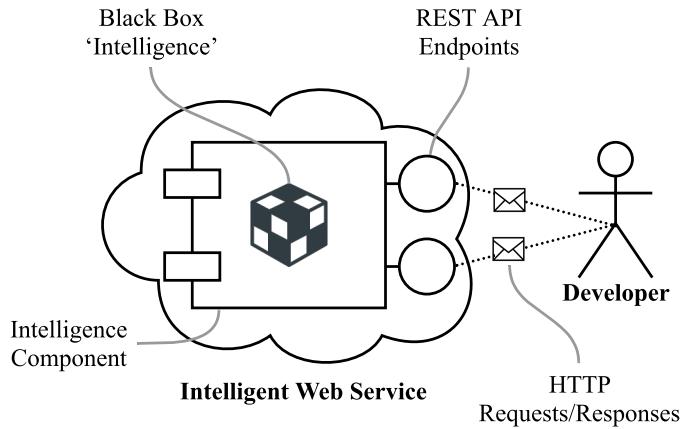


Figure 1.4: Overview of intelligent web services (IWSs).

⁹⁶ (typically) available as a cloud-based Platform as a Service (PaaS).^{6,7} Developers
⁹⁷ send a given request to analyse a specific piece of data (e.g., an image, body of text,
⁹⁸ audio file etc.) and receive some intelligence on the data (e.g., object detection, text
⁹⁹ sentiment, transcription of audio) in addition to an associated *confidence* value that
¹⁰⁰ represents the likelihood of that result. This is typically serialised as a JSON/XML
¹⁰¹ response object.

☞ *Within this thesis, we scope our investigation to a mature subset of IWSs that provide computer vision intelligence [394, 397, 410, 411, 412, 418, 422, 431, 432, 434, 436, 484, 485]. For the context of this thesis, we will refer to such services as CVSS.*

¹⁰² 1.2 Motivating Scenarios

¹⁰³ The market for computer vision services (CVSS) is expanding (Table 1.1) with a
¹⁰⁴ corresponding interest from developers (Figure 1.1). These services are inherently
¹⁰⁵ probabilistic in their behaviour, in that the end-points always return with a response
¹⁰⁶ with a probability. This is unlike a typical API that would return a response (*without*
¹⁰⁷ a probability) or an error. If developers do not fully understand the nature of these

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.

¹⁰⁸ services when integrating with them, there is an impact on the quality of software
¹⁰⁹ they create.

¹¹⁰ To illustrate the context of use, we present the two scenarios of varying risk:
¹¹¹ (i) a fictional software developer, named Tom, who wishes to develop an inherently
¹¹² low-risk photo labelling application for his friends and family; and (ii) a high-risk
¹¹³ cancer clinical decision support system (CDSS) that uses patient scans to recom-
¹¹⁴ mend if surgeons should send their patients to surgery. Both describe scenarios
¹¹⁵ where AI-infused components has an impact to end-users when the software engi-
¹¹⁶ neers developing with them misunderstand the nuances of ML, ultimately affecting
¹¹⁷ external quality. Moreover, when developers lack a comprehension, this hinders
¹¹⁸ their productivity and understanding/appreciation of AI-based components.

¹¹⁹ 1.2.1 Low Risk Motivating Scenario

¹²⁰ Tom wants to develop a social media photo-sharing app on iOS and Android, *Photo-*
¹²¹ *Sharer*, that analyses photos taken on smartphones. Tom wants the app to categorise
¹²² photos into scenes (e.g., day vs. night, landscape vs. indoors), generate brief de-
¹²³ scriptions of each photo, and catalogue photos of his friends and common objects
¹²⁴ (e.g., photos with his Border Collie dog, photos taken on a beach on a sunny day with
¹²⁵ his partner). His app will shares this analysed photo intelligence with his friends on
¹²⁶ a social-media platform, where his friends can search and view the photos.

¹²⁷ Instead of building a computer vision engine from scratch, which takes too much
¹²⁸ time and effort, Tom thinks he can achieve this using one of the common CVSs. Tom
¹²⁹ comes from a typical software engineering background and has insufficient knowl-
¹³⁰ edge of key computer vision terminology and no understanding of its underlying
¹³¹ techniques. However, inspired by easily accessible cloud APIs that offer computer
¹³² vision analysis, he chooses to use these. Built upon his experience of using other
¹³³ similar cloud services, he decides on one of the CVS APIs, and expects a static result
¹³⁴ always and consistency between similar APIs. Analogously, when Tom invokes the
¹³⁵ iOS Swift substring method "doggy".prefix(3), he expects it to be consistent
¹³⁶ with the Android Java equivalent "doggy".substring(0, 2). Consistent, here,
¹³⁷ means two things: (i) that calling substring or prefix on 'dog' will *always*
¹³⁸ return in the same way every time he invokes the method; and (ii) that the result is
¹³⁹ *always* 'dog' regardless of the programming language or string library used, given
¹⁴⁰ the deterministic nature of the 'substring' construct (i.e., results for substring are
¹⁴¹ API-agnostic).

¹⁴² More concretely, in Table 1.4, we illustrate how three (anonymised) CVS
¹⁴³ providers fail to provide similar consistency to that of the substring example above.
¹⁴⁴ If Tom uploads a photo of a border collie⁸ to three different providers in August
¹⁴⁵ 2018 and January 2019, he would find that each provider is different in both the vo-
¹⁴⁶ cabulary used between. The confidence values and labels within the *same* provider
¹⁴⁷ varies within a matter of five months. The evolution of the confidence changes is not
¹⁴⁸ explicitly documented by the providers (i.e., when the models change) nor do they
¹⁴⁹ document what confidence means. Service providers use a tautological nature when

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

Table 1.4: 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

150 defining what the confidence values are (as presented in the API documentation)
151 provides no insight for Tom to understand why there was a change in confidence,
152 which we show in Table 1.5, unless he *knows* that the underlying models change with
153 them. Furthermore, they do not provide detailed understanding on how to select a
154 threshold cut-off for a confidence value. Therefore, he's left with no understanding
155 on how best to tune for image classification in this instance. The deterministic prob-
156 lem of a substring compared to the nondeterministic nature of the IWS is, therefore,
157 non-trivial.

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

- 161 • What does ‘confidence’ mean?
- 162 • Which confidence is acceptable in this scenario?
- 163 • Are these APIs consistent in how they respond?
- 164 • Are the responses in APIs static and deterministic?
- 165 • Would a combination of multiple CVS APIs improve the response?
- 166 • How does he know when there is a defect in the response? How can he report
167 it?
- 168 • How does he know what labels the API knows, and what labels it doesn't?
- 169 • How does it describe his photos and detect the faces?
- 170 • Does he understand that the API uses a machine learnt model? Does he know
171 what a ML model is?
- 172 • Does he know when models update? What is the release cycle?

Table 1.5: Tautological definitions of ‘confidence’ found in the API documentation of three common CVS providers.

API Provider	Definition(s) of Confidence
Provider A	<p>“Score is the confidence score, which ranges from 0 (no confidence) to 1 (very high confidence).” [420]</p> <p>“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].” [421]</p> <p>“The overall score of the result. Range [0, 1]” [421]</p>
Provider B	<p>“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.” [437]</p> <p>“The level of confidence the service has in the caption.” [435]</p>
Provider C	<p>“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.” [395]</p> <p>“[Provider C] also provide[s] a percentage score for how much confidence [Provider C] has in the accuracy of each detected label.” [396]</p>

173 Although Tom generally anticipates these CVSs to not be perfect, he has no
 174 prior benchmark to guide him on what to expect. The imperfections appear to be
 175 low-risk, but may become socially awkward when in use; for instance, if Tom’s
 176 friends have low self-esteem and use the app, they may be sensitive to the app not
 177 identifying them or mislabelling them. Privacy issues come into play especially
 178 if certain friends have access to certain photos that they are (supposedly) in; e.g.,
 179 photos from a holiday with Tom and his partner, however if the API identifies Tom’s
 180 partner as a work colleague, Tom’s partner’s privacy is at risk.

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

184 1.2.2 High Risk Motivating Scenario

185 Recent studies in the oncology domain have used deep-learning convolutional neural
 186 networks (CNNs) to detect region of interests (ROIs) in image scans of tissue (e.g.,
 187 [33, 148, 221]), flagging these regions for doctors to review. Trials of such algorithms
 188 have been able to accurately detect cancer at higher rates than humans, and thus
 189 incorporating such capabilities into a CDSS is closer within reach. Studies have
 190 suggested these systems may erode a practitioner’s independent decision-making

[74, 176] due to over-reliance; therefore the risks in developing CDSSs powered by IWSs become paramount.

In Figure 1.5 we present a context diagram for a fictional CDSS named *CancerAssist*. A team of busy pathologists utilise CancerAssist to review patient lymph node scans and discuss and recommend, on consensus, if the patient requires an operation. When the team makes a consensus, the lead pathologist enters the verdict into CancerAssist—running passively in the background—to ensure there is no oversight in the team’s discussions. When a conflict exists between the team’s verdict and CancerAssist’s verdict, the system produces the scan with ROIs it thinks the team should review. Where the team overrides the output of CancerAssist, this reinforces CancerAssist’s internal model as a human-in-the-loop (HITL) learning process.

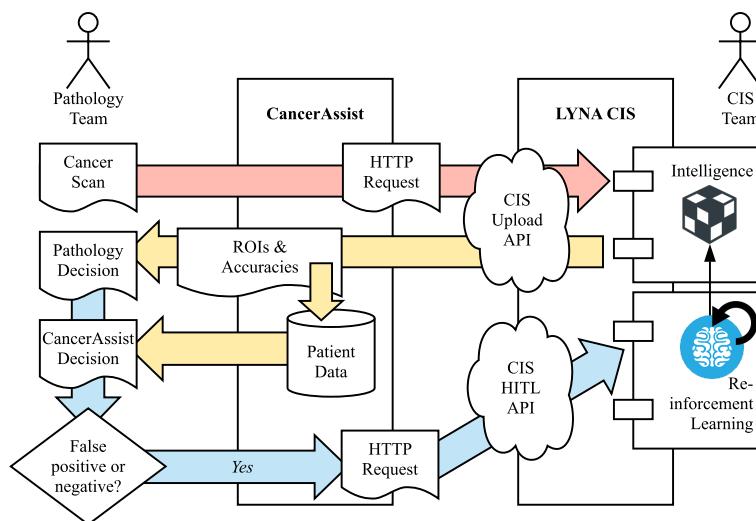


Figure 1.5: CancerAssist Context Diagram. **Key:** Red Arrows = Scan Input; Yellow Arrows = Decision Output; Blue Arrows = HITL Feedback Input.

Powering CancerAssist is Google AI’s Lymph Node Assistant (LYNA) [221], a CNN based on the Inception-v3 model [204, 346]. To provide intelligence to CancerAssist, the development team decide to host LYNA as an IWS using a cloud-based PaaS solution. Thus, CancerAssist provides API endpoints integrated with patient data and medical history, which produces the verdict. In the case of a positive verdict, CancerAssist highlights the relevant ROIs found with their respective bounding boxes and their respective cancer detection accuracies.

The developer of CancerAssist has no interaction with the Data Science team maintaining the LYNA IWS. As a result, they are unaware when updates to the model occur, nor do they know what training data they provide to test their system. The default assumptions are that the training data used to power the intelligence is near-perfect for universal situations; i.e., the algorithm chosen is the correct one for every assessable ontology tests in the given use case of CancerAssist. Thus, unlike deterministic systems—where the developer can manually test and validate the outcomes of the APIs—this is impossible for non-deterministic systems such

²¹⁸ as CancerAssist and its underlying IWS. The ramifications of not being able to test
²¹⁹ such a system and putting it out into production may prove fatal to patients.

²²⁰ Certain questions in the production of CancerAssist and its use of an IWS may
²²¹ come into mind:

- ²²² • When is the model updated and how do the IWS team communicate these
²²³ updates?
- ²²⁴ • What benchmark test set of data ensures that the changed model doesn't affect
²²⁵ other results?
- ²²⁶ • Are assumptions made by the IWS team who train the model correct?

²²⁷ Thus, to improve communication between developers and IWS providers, devel-
²²⁸ opers require enhanced documentation, additional metadata, and guidance tooling.

²²⁹ 1.3 Research Motivation

²³⁰ Evermore applications are considering IWSs as demonstrated by ubiquitous exam-
²³¹ ples: aiding the vision-impaired [94, 298], accounting [229], data analytics [174],
²³² and student education [100]. Our motivating examples illustrate impact on de-
²³³ velopers when CVSSs encapsulate assumptions and are poorly documented. Such
²³⁴ components are accessible through APIs consisting of ‘black box’ intelligence (Fig-
²³⁵ ure 1.4).⁹ ML models are inherently probabilistic and stochastic, contributing to
²³⁶ four critical issues for developers that motivate this research work: (i) communica-
²³⁷ tion of outputs (as probabilities), (ii) evolution of datasets, (iii) selecting appropriate
²³⁸ decision boundaries, and (iv) the clarity of documentation that address items i–iii.
²³⁹ We detail these four issues in the following subsections.

²⁴⁰ Ultimately, these four issues present major threats to software reliability if left
²⁴¹ unresolved. Given that such substantiative software engineering principles on re-
²⁴² liability, versioning and quality are under-investigated within the context of IWSs,
²⁴³ we aim to explore guidance from the software engineering literature to investigate
²⁴⁴ what aspects in the development lifecycle could aide in mitigating these issues when
²⁴⁵ developing using components that abstract ML, such as IWSs.

²⁴⁶ 1.3.1 Outputs are Probabilities

²⁴⁷ There is little room for certainty in these results as the insight is purely statistical
²⁴⁸ and associational [280] against its training dataset. **The interface between AI-**
²⁴⁹ **components and traditional software components is non-trivial when developers**
²⁵⁰ **do not appreciate the nuances, or use the anchors of libraries and components**
²⁵¹ **that have a more traditional behaviour [187, 246, 357, 362].** However, CVSSs
²⁵² return the *probability* that a particular object exists in an input images’ pixels via
²⁵³ confidence values. As an example, consider simple arithmetic representations (e.g.,

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 [17, 64] to describe “systems whose internal mechanisms are not fully open to inspection” [17].

254 $2 + 2 = 4$). The deterministic mindset suggests that the result will *always* be
255 4. However, the non-deterministic (data-driven) mindset suggests that results are
256 probable: target output (*exactly* 4) and the output inferred (*a likelihood of* 4) matches
257 as a probable percentage (or as an error where it does not match).¹⁰ Instead of an
258 exact output, there is a *probabilistic* result: $2 + 2$ *may* equal 4 to a confidence of n .
259 Thus, for a more certain (though not fully certain) distribution of overall confidence
260 returned from the service, a developer must treat the problem stochastically by
261 testing this case hundreds if not thousands of times to find a richer interpretation of
262 the inference made and ensure reliability in its outcome.

263 1.3.2 Evolution of Datasets

264 Traditional software engineering principles advocate for software systems to be ver-
265 sioned upon substantial change. Unfortunately, endpoints are not versioned [88]. In
266 the context of computer vision, new labels may be introduced or dropped, confidence
267 values may differ, entire ontologies or specific training parameters may change, but
268 we hypothesise that is not effectively communicated to developers. Broadly speak-
269 ing, this can be attributed to a dichotomy of release cycles from the data science and
270 software engineering communities: the data science iterations and work by which
271 new models are trained and released runs at a faster cycle than the maintenance
272 cycle of traditional software engineering. Thus we see cloud vendors integrating
273 model changes without the *need* to update the API version unless substantial code or
274 schema changes are also introduced—the nuance changes in the internal model does
275 not warrant a shift in the API itself, and therefore the version shift in a new model
276 does not always propagate to a version shift in the API endpoint. As demonstrated
277 in Table 1.4, whatever input is uploaded at one time may not necessarily be the same
278 when uploaded at a later time. This again contrasts the rule-driven mindset, where
279 $2 + 2$ *always* equals 4. Therefore, in addition to the certainty of a result in a single
280 instance, the certainty of a result in *multiple instances* may differ with time, which
281 again impacts on the developers notion of reliable software. Currently, it is impos-
282 sible to invoke requests specific to a particular model that was trained at a particular
283 date in time, and therefore developers need to consider how evolutionary changes of
284 the services may impact their solutions *in production*. Again, whether there is any
285 noticeable behavioural changes from these changes is dependent on the context of
286 the problem domain—unless developers benchmark these changes against their own
287 domain-specific dataset and frequently check their selected service against such a
288 dataset, there is no way of knowing if substantive errors have been introduced.

289 1.3.3 Selecting Appropriate Decision Boundaries

290 As the only response from these computer vision classifiers are a label and confidence
291 value; **the decision boundaries needs to always be appropriately considered by**
292 **client code for each use case and each model selected.** The external quality of

293¹⁰Blake et al. [44] produces a multi-layer perceptron neural network performing arithmetic repre-
294 sentation.

such software needs to consider reliability in the case of thresholding confidence values—that is whether the inference has an appropriate level of confidence to justify a predicted (and reliable) result to end-users. Selecting this confidence threshold is non-trivial; a ML course from Google suggests that “it is tempting to assume that [a] classification threshold should always be 0.5, but thresholds are problem-dependent, and are therefore values that you must tune.” [143]. Approaches to turning these values are considered for data scientists, but are not yet well-understood for application developers with little appreciation of the nuances of ML.

1.3.4 Documentation of the above concerns

Similarly, developers should consider the internal quality of building AI-first software. Reliable API usability and documentation advocate for the accuracy, consistency and completeness of APIs and their documentation [285, 304] and providers should consider mismatches between a developer’s conceptual knowledge of the API its implementation [198]. **Unreliable APIs ultimately hinder developer performance and thus reduces productivity**, in addition to producing potentially unreliable software where documentation is not well-understood (or clear to the developer).

1.4 Research Goals

This thesis aims to investigate and better understand the nature of cloud-based computer vision services (CVSs)¹¹ as a concrete exemplar of intelligent web services (IWSs). We identify the maturity, viability and risks of CVSs through the anchoring perspective of *reliability* that affects the internal and external quality of software. We adopt the McCall [232] and Boehm [46] interpretations of reliability via the sub-characteristics of a service’s *consistency* and *robustness* (or fault/error tolerance), and the *completeness*¹² of its documentation. (A detailed discussion is further provided in Section 2.1.) This thesis explores and contributes towards *four* key facets regarding reliability in CVS usage and the completeness of its associated documentation. We formulate four primary research questions (RQs), based on both empirical and non-empirical software engineering methodology [242], further discussed in Chapter 3.

Firstly, we investigate adverse implications that arise when using CVSs that affects consistency and robustness (**Chapter 4**). We show how CVSs have a non-deterministic runtime behaviour and evolve with unintended and non-trivial consequences to developers. We demonstrate that these services have inconsistent behaviour despite offering the same functionality and pose evolution risk that effects robustness of consuming applications when responses change given the same (consistent) inputs.

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.

³³⁰ Formally, we structure the following RQs:

?

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

RQ1.1. What is their runtime behaviour?

RQ1.2. What is their evolution profile?

³³¹ Secondly, we investigate the reliability of the documentation these services offer through the lenses of its completeness. We collate prior knowledge of good ³³² API documentation and assess the efficacy of such knowledge against practitioners ³³³ (**Chapter 8**). We show that these service's behaviour and evolution is not ³³⁴ reliably documented adequately against this knowledge. Formally, we develop the ³³⁵ following RQs: ³³⁶

?

RQ2. Are CVS APIs sufficiently documented?

RQ2.1. What API documentation artefacts compromise 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?

³³⁷ Thirdly, we investigate how software developers approach using these services ³³⁸ and directly assess developer pain-points resulting from the nature of CVSs and ³³⁹ their documentation (**Chapter 5**). We show that there is a statistically significant ³⁴⁰ difference in these complaints when contrasted against more established software ³⁴¹ engineering domains (such as web or mobile development) as expressed as questions ³⁴² asked on Stack Overflow. We provide a number of exploratory avenues for ³⁴³ researchers, educators, software engineers and IWS providers to alleviate these ³⁴⁴ complaints based on this analysis. Further, using a data set consisting of 1,245 Stack ³⁴⁵ Overflow questions, we explore the emotional state of developers to understand ³⁴⁶ which aspects (i.e., pain-points) developers are most frustrated with (**Chapter 6**) ³⁴⁷ and the types of traps developers can fall into when substantial documentation is not ³⁴⁸ provided for specific ML models (**Chapter 7**). We formulate the following RQs:

?

RQ3. Are CVSs more misunderstood than conventional software engineering 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?

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

358 **② RQ4. What strategies can developers employ to integrate their appli-
cations with CVSs while preserving robustness and reliability?**

1.5 Research Methodology

359 This thesis employs a mixed-methods approach using the concurrent triangulation
360 strategy [57, 231]. The research presented consists of both empirical and non-
361 empirical research design. This section provides a high-level overview of the re-
362 search methodology within this thesis. Further details are provided in Section 1.7
363 and Chapter 3.

364 Firstly, RQ1–RQ3 are all empirical, knowledge-based questions [109, 238] that
365 aim to provide the software engineering community with a greater understanding
366 of the phenomena surrounding CVSs from three perspectives: the nature of the ser-
367 vices themselves, how developers perceive these services and how service providers
368 can improve these services. We answer RQ1 using a longitudinal experiment that
369 assesses both the services’ responses and associated documentation (complement-
370 ing RQ2.2). We adopt qualitative and quantitative data collection; specifically (i)
371 structured observations to quantitatively analyse the results over time, and (ii) docu-
372 mentary research methods to inspect service documentation. Secondly, we perform
373 systematic mapping study following the guidelines of Kitchenham and Charters
374 [194] and Petersen et al. [282] to better understand how API documentation of these
375 services can be improved (i.e., more complete), which targets RQ2. Based on the
376 findings from this study, we use a systematic taxonomy development methodol-
377 ogy specifically targeted toward software engineering [360] that structures scattered
378 API documentation knowledge into a taxonomy. We then validate this taxonomy
379 against practitioners using survey research, using a survey instrument inspired by
380 Brooke’s well-established System Usability Scale [61] and contextualising it within
381 API documentation utility, which answers RQ3.3. To answer RQ2.2, we perform
382 an empirical application of the taxonomy to three CVSs, and therefore assess where
383 improvements can be made. Thirdly, we adopt field survey research using repository
384 mining of developer discussion forums (i.e., Stack Overflow) to answer RQ3, and
385 classify these using both manual and automated techniques.

386 The second aspect of our research design involves non-empirical research, which
387 explores a design-based question [242] to answer RQ4. As the answers to our

388 first three RQs establish a greater understanding of the nature behind CVSs from
389 various perspectives, the strategies we design in RQ4 aims at designing more reliable
390 integration methods so that developers can better use these cloud-based services in
391 their applications.

392 1.6 Thesis Organisation

393 We organise the thesis into four parts. **Part I (The Preface)** includes introductory,
394 background and methodology chapters. This is a *PhD by Publication*, and
395 **Part II (Publications)** comprises of eight publications resulting from this work over
396 Chapters 4 to 11; publications are included verbatim except for terminology and for-
397 matting changes to better fit the suitability of a coherent thesis. **Part III (The Post-**
398 **face)** includes the conclusion and future works chapter, as well as a list of academic
399 studies and online artefacts referenced within the thesis. **Part IV (Appendices)** in-
400 cludes all supplementary material, including mandatory authorship statements and
401 ethics approval. Details of each chapter following this introductory chapter are
402 provided in the following section.

403 1.6.1 Part I: Preface

404 1.6.1.1 Chapter 2: Background

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

410 1.6.1.2 Chapter 3: Research Methodology

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

415 1.6.2 Part II: Publications

416 1.6.2.1 Chapter 4: Exploring the nature of CVSs

417 This chapter was presented at the 2019 **International Conference on Software**
418 **Maintenance and Evolution (ICSME)** [88]. We describe an 11-month longitudi-
419 nal experiment assessing the behavioural (run-time) issues of three popular CVSs:
420 Google Cloud Vision [422], Amazon Rekognition [397] and Azure Computer Vi-
421 sion [436]. By using three different data sets—two of which we curate as additional
422 contributions—we demonstrate how the services are inconsistent amongst each other

⁴²³ and within themselves. This study answers RQ1: Despite presenting conceptually-
⁴²⁴ similar functionality, each service behaves and produces slightly varied (inconsistent)
⁴²⁵ results and demonstrates non-deterministic runtime behaviour. We discuss potential
⁴²⁶ evolution risks to consumers of such services as the services provide non-static
⁴²⁷ outputs for the same inputs, thereby having significant impact to the robustness of
⁴²⁸ consuming applications. Further details in the study include a brief assessment into
⁴²⁹ the lack of sufficient detail of these concerns in their documentation.

⁴³⁰ 1.6.2.2 *Chapter 5: Understanding developer struggles when using CVSs*

⁴³¹ This chapter was presented at the **2020 International Conference on Software**
⁴³² **Engineering (ICSE)** [91]. We conduct a mining study of 1,425 Stack Overflow
⁴³³ questions that provide indications of the types frustrations that developers face when
⁴³⁴ integrating CVSs into their applications. To gather what their pain-points are, we use
⁴³⁵ two classification taxonomies that also use Stack Overflow to understand generalised
⁴³⁶ and documentation-specific pain-points in mature software engineering domains.
⁴³⁷ This study answers RQ3 in detail and provides a validation to our motivation of
⁴³⁸ RQ2: we validate that the *completeness* of current CVS API documentation is a
⁴³⁹ main concern for developers and there is insufficient explanation into the errors
⁴⁴⁰ and limitations of the service. We find that the documentation does not adequately
⁴⁴¹ cover all aspects of the technical domain. In terms of integrating with the service,
⁴⁴² developers struggle most with simple errors and ways in which to use the APIs; this
⁴⁴³ is in stark contrast to mature software domains. Our interpretation is that developers
⁴⁴⁴ fail to understand the IWS lifecycle and the ‘whole’ system that wraps such services.
⁴⁴⁵ We also interpret that developers have a shallower understanding of the core issues
⁴⁴⁶ within CVSs (likely due to the nuances of ML as suggested in a discussion in the
⁴⁴⁷ paper), which warrants an avenue for future work in software engineering education.

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

⁴⁴⁹ This chapter has been published as a technical report pre-print on arXiv and an
⁴⁵⁰ extended version is **in review** for submission to the **2021 International Workshop**
⁴⁵¹ **on Emotion Awareness in Software Engineering (SEmotion)** [86]. In this work,
⁴⁵² we use our dataset consisting of the 1,425 Stack Overflow questions from [91] to
⁴⁵³ interpret the breakdown of emotions developers express per classification of pain-
⁴⁵⁴ points conducted in Chapter 5. We find that the distribution of various emotions
⁴⁵⁵ differ per question type, and developers are most frustrated when the expectations
⁴⁵⁶ of a CVS does not match the reality of what these services actually provide, which
⁴⁵⁷ shapes our answer for RQ3.2 and thus RQ3.

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

⁴⁵⁹ This chapter is **in review** for the **2021 International Conference on Advanced**
⁴⁶⁰ **Information Systems Engineering (CAiSE)** [144]. This work presents a deeper
⁴⁶¹ investigation into the classification model used within Chapter 6 to better interpret the
⁴⁶² automation effort we conducted, thereby highlighting valuable lessons we learnt from

Table 1.6: 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
[88]	35 th International Conference on Software Maintenance and Evolution	ICSME	A	Sep 2019	Chapter 4	RQ1
[87]	13 th International Symposium on Empirical Software Engineering and Measurement	ESEM	A	Sep 2019	Excluded ¹⁵	RQ2.1
[91]	42 nd International Conference on Software Engineering	ICSE	A*	Jun 2020	Chapter 5	RQ3
[86]	6 th International Workshop on Emotion Awareness in Software Engineering ¹⁶	SEmotion	A*	<i>In Review</i>	Chapter 6	RQ3.2
[144]	33 rd International Conference on Advanced Information Systems Engineering	CAiSE	A	<i>In Review</i>	Chapter 7	RQ3.2
[92]	IEEE Transactions on Software Engineering	TSE	Q1	Dec 2020	Chapter 8	RQ2
[265]	13 th International Conference on Web Engineering	ICWE	B	Apr 2019	Chapter 9	RQ4
[89]	28 th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	FSE(d) ¹⁷	A*	Nov 2020	Chapter 10	RQ4
[90]	28 th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	FSE	A*	Nov 2020	Chapter 11	RQ4

¹³Measured as at Jan 2020 using CORE Conference (<http://www.core.edu.au/conference-portal>) and Scimago Rankings (<https://scimagojr.com/>).¹⁴Date of publication, if applicable.¹⁵The extended version of this conference proceeding is provided in [92], Chapter 8.¹⁶An ICSE 2021 workshop.¹⁷We abbreviate this with an added ‘d’ (for the demonstrations track) to distinguish this paper from our full FSE 2020 paper.

⁴⁶³ performing this exercise. Specifically, we find that the classification model we used
⁴⁶⁴ in this exercise presented substantial data imbalance, which presented unexpected
⁴⁶⁵ results (namely, a high level of posts that showed the emotion, ‘love’). We identify
⁴⁶⁶ how novel documentation tooling such as model cards [245] or datasheets [133]
⁴⁶⁷ could have identified risks to our study earlier, and make suggestions needed into
⁴⁶⁸ future documentation efforts. This work presents complementary results to RQ2 to
⁴⁶⁹ help propose which documentation elements ML models (and thus IWSs) should
⁴⁷⁰ provide before diving ‘straight in’.

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

⁴⁷² This chapter was accepted as a paper at the **2019 International Symposium on**
⁴⁷³ **Empirical Software Engineering and Measurement (ESEM)** [91]. The extended
⁴⁷⁴ version of this chapter was published in the **IEEE Transactions on Software En-**
⁴⁷⁵ **gineering (TSE)** [92]. To understand where to improve CVS documentation, we
⁴⁷⁶ first need to investigate *what* makes a good API document. This short paper initially
⁴⁷⁷ answered one aspect of RQ2.1: the extent by which *academic literature* has studied
⁴⁷⁸ various API documentation artefacts. By conducting an systematic mapping study
⁴⁷⁹ resulting in 21 primary studies, we systematically develop a taxonomy that combines
⁴⁸⁰ documentation artefacts studied in scattered work into a structured framework of 5
⁴⁸¹ dimensions and 34 weighted categorisations. We then extend this work by trian-
⁴⁸² gulating the taxonomy with opinions from developers using a survey to assess the
⁴⁸³ efficacy of these artefacts (thereby answering the second aspect of RQ2.1). From
⁴⁸⁴ this, we assess the how well CVS providers document their APIs via a heuristic
⁴⁸⁵ validation of the taxonomy, using the three services from the ICSME publication
⁴⁸⁶ to make recommendations where documentation should be more complete, thereby
⁴⁸⁷ answering RQ2.2 (and thus RQ2).

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

⁴⁸⁹ This chapter was presented at the **2019 International Conference on Web Engi-**
⁴⁹⁰ **neering (ICWE)** [265]. Early exploration of CVSs showed that multiple services
⁴⁹¹ use vastly different ontologies for the same input. As an initial strategy to improve
⁴⁹² the reliability of these services, we explored if merging multiple responses using
⁴⁹³ WordNet [244] and a novel label merging algorithm based on the proportional rep-
⁴⁹⁴ resentation approach used in political voting could make any improvements. While
⁴⁹⁵ this approach resulted in a modest improvement to reliability, it did not consider to
⁴⁹⁶ the evolution issues or developer pain-points we later identified.

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

⁴⁹⁸ This chapter was presented at the demonstrations track of the **2020 Joint European**
⁴⁹⁹ **Software Engineering Conference and Symposium on the Foundations of Soft-**
⁵⁰⁰ **ware Engineering (ESEC/FSE)** [89]. When integrating with a CVS, developers
⁵⁰¹ need to select an appropriate confidence threshold suited to their use case and deter-
⁵⁰² mine whether a decision should be made. An issue, however, is that these CVSs are

503 not calibrated to the specific problem-domain datasets and it is difficult for software
504 developers to determine an appropriate confidence threshold on their problem do-
505 main. This tool presents a workflow and supporting tool for application developers
506 to select decision thresholds suited to their domain that—unlike existing tooling—is
507 designed to be used in pre-development, pre-release and production. This tooling
508 forms part of a solution to RQ4 for developers to maintain robustness and reliability
509 in their systems.

510 **1.6.2.8 Chapter 11: Developing a CVS integration architecture**

511 This chapter was presented at the **2020 Joint European Software Engineering**
512 **Conference and Symposium on the Foundations of Software Engineering (ES-**
513 **EC/FSE)** [90]. Based on the findings, we propose a set of new service error codes
514 for describing the empirically observed error conditions of IWS based on our find-
515 ings in Chapter 4. To achieve this, we propose a proxy server intermediary that lies
516 between a client application and a IWS; the proxy server tactic is designed to return
517 these error codes when substantial evolution occurs against a benchmark dataset that
518 represents the application domain context (similar to that proposed in Chapter 10).
519 A technical evaluation of our implementation of this architecture identifies 1,054
520 cases of substantial evolution in confidence values and 2,461 cases of evolution in
521 the response label sets when 331 images were sent to a CVS.

522 **1.6.3 Part III: Postface**

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

529 **1.6.4 Part IV: Appendices**

530 Chapter A thru Chapter E are appendices. Chapter A provides additional material
531 referenced within this thesis but not provided in the body. The source code for the
532 reference architecture described in Chapter 11 is reproduced in Chapter B. The sup-
533 plementary materials published with Chapter 8 are reproduced in Chapter C, which
534 also describes the list of primary sources arising in the systematic mapping study
535 we conducted. We provide mandatory coauthor declaration forms describing the
536 contribution breakdown for each publication within Chapter D. Chapter E contains
537 copies of the ethics clearance for various experiments within this thesis.

538 **1.7 Research Contributions**

539 The outcomes of answering the four primary research questions elaborated in Sec-
540 tion 1.4 shapes three primary contributions this thesis offers to software engineering

⁵⁴¹ knowledge:

- ⁵⁴² • An **improved understanding in the landscape of CVSs**, with respect to their runtime behaviour and evolutionary profiles.
- ⁵⁴³
- ⁵⁴⁴ • A novel **service integration architecture** that helps developers with integrating their applications with CVSs.
- ⁵⁴⁵
- ⁵⁴⁶ • A **key list of attributes that should be documented**, to assist CVS providers to better document their services.
- ⁵⁴⁷

⁵⁴⁸ In this section, we detail how each publication forms a coherent body of work and how each publication relates to the primary contributions made.

⁵⁴⁹ After our exploratory analysis on the nature of CVSs (Chapter 4), we proposed two sets of recommendations targeted towards two stakeholders: (i) the service *consumers* (i.e., application developers) and (ii) the service *providers*. Our subsequent publications arose as a two-fold investigation to develop two strategies in which developers and providers can, respectively, (i) better integrate these intelligent components into their applications, and (ii) how these services can be better documented. Table 1.6 provides a tabulated form of the publications and research questions addressed within this thesis; for ease of reference, we refer to the publications in within this section in their abbreviated form as listed in Table 1.6. We also provide abbreviations for easier reference in this section. A high-level overview of the cohesiveness of our publications is provided in Figure 1.6.

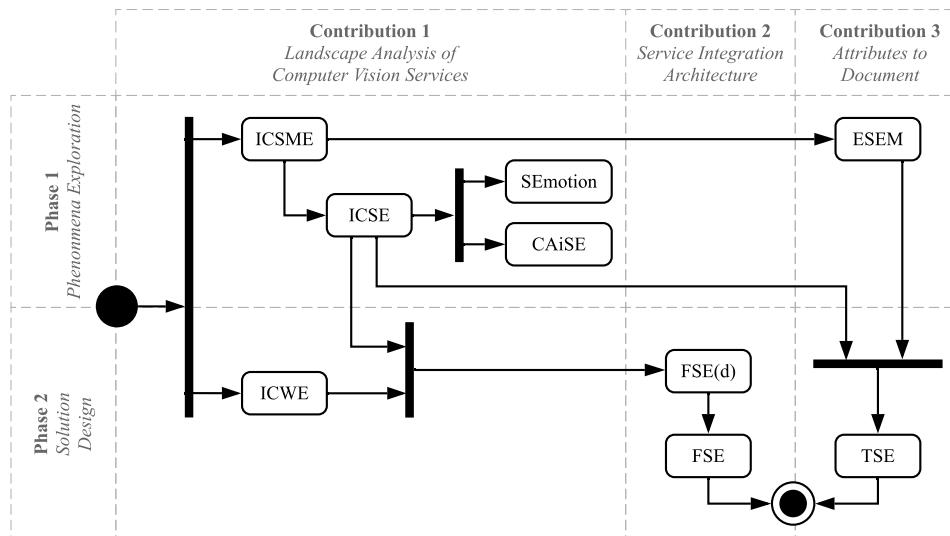


Figure 1.6: Activity diagram of the coherency of our publications, how our research was conducted, and relevant connections between publications. Our two-phase structure initial phenomena exploration and a proposed solutions to issues identified from the exploration. We map the contributions within each publication to the three primary contributions of the thesis. Acronyms of each publication are provided in Table 1.6.

561 1.7.1 Contribution 1: Landscape Analysis & Preliminary Solutions

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

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

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

588 1.7.2 Contribution 2: Improving Documentation Attributes

589 As mentioned, our ICSME paper found that evolutionary and non-deterministic
590 behavioural profile of are not adequately documented in pre-trained ML model APIs
591 documentation, and further developers find this frustrating (Chapter 6) and potential
592 issues can arise as a result (Chapter 7). A recommendation concluding from this
593 work was that service providers should improve their documentation, however there
594 lacked a strategy by which they could do this, and our hypotheses that developers
595 were actually frustrated by this lack of communication was yet to be tested. This led
596 to two follow-up further investigations as presented in our ICSE and ESEM papers.

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

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

618 1.7.3 Contribution 3: Service Integration Architecture

619 Two recommendations from our ICSME study encouraged developers to test their
620 applications with a representative ontology for their problem domain and to incorpo-
621 rate a specialised testing and monitoring techniques into their workflow. Strategies
622 on *how* to achieve this were explored in later studies. Following a similar approach
623 to our solution of improved API documentation, we validated the substantiveness of
624 our recommendations using our mining study of Stack Overflow (our ICSE paper)
625 to help inform us of generalised issues developers face whilst integrating CVSs into
626 their applications. To achieve this, we used a Stack Overflow post classification tax-
627 onomy proposed by Beyer et al. [40] into seven categories, where 28.9% and 20.37%
628 of posts asked issues regarding how to use the CVS API and conceptual issues be-
629 hind CVSs, respectively. Developers presented an insufficient understanding of the
630 non-deterministic runtime behaviour, functional capability, and limitations of these
631 services and are not aware of key computer vision terminology. When contrasted
632 to more conventional domains such as mobile-app development, the spread of these
633 issues vary substantially.

634 We proposed two technical solutions in our two FSE papers to help alleviate
635 this issue. Firstly, our FSE demonstrations paper—FSE(d) for short—provides a
636 workflow for developers to better select an appropriate confidence threshold, and
637 thus decision boundary, calibrated for their particular use case. In our ESEC/FSE
638 paper, we provide a reference architecture for developers to guard against the non-
639 deterministic issues that may ‘leak’ into their applications. This architecture tactic
640 proposes a client-server intermediary proxy server, similar to the style proposed in
641 our ICWE paper. However, unlike the ICWE paper that uses proportional repre-
642 sentation approach to modify multiple sources, our FSE paper proposes a guarded
643 relay, whereby a single service is used, and the proxy server maintains a lifecycle to
644 monitor evolution issues identified in ICSME and should be benchmarked against
645 the developer’s dataset (i.e., against the particular application domain) as suggested
646 in FSE(d). For robust component composition, this architecture tactic handles four

647 key requirements: (i) it clearly defines erroneous conditions that occur when evo-
648 lution occurs in CVSs; (ii) it notifies of behavioural changes in the service; (iii) it
649 monitors the service for change and substantial impact this may have to the client
650 application; and (iv) is flexible enough to be implemented and adaptable to any client
651 application or specific intelligent service to facilitate reuse. Both FSE papers serve
652 as two primary contributions to RQ4.

CHAPTER 2

653

654

655

Background

656

657 In Chapter 1, we defined a common set of (artificial) intelligence-based cloud ser-
658 vices that we label intelligent web services (IWSs). Specifically, we scope the
659 primary body of this study’s work on computer vision services (CVSs) (e.g., Google
660 Cloud Vision [422], AWS Rekognition [397], Azure Computer Vision [436], Wat-
661 son Visual Recognition [432] etc.). We claim developers have not yet internalised
662 the nuances of working with components that have a probabilistic behaviour ($2 + 2$
663 *always equals 4*) whereas an IWS’s ‘intelligence’ component (a black box) returns
664 probabilistic results ($2 + 2$ *might equal 4 with a confidence of 95%*). Thus, there is a
665 mindset mismatch between probabilistic results (from the API provider) and results
666 interpreted with certainty (from the API consumer).

667 What affect does this anchor mismatch have on the developer’s approach towards
668 building probabilistic software? What can we learn from common software engi-
669 neering practices (e.g., [288, 335]) that apply to resolve this mismatch and thereby
670 improve quality, such as verification & validation (V&V)? Chiefly, we consider this
671 question around three lenses of software engineering: creating an IWS, using an
672 IWS, and the nature of IWSs themselves.

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

677 **(1) The development of the IWS.** We investigate the internal quality attributes
678 of creating an IWS from the IWS *provider’s* perspective. That is, we ask if
679 existing verification techniques are sufficient enough to ensure that the IWS
680 being developed actually satisfies the IWS consumer’s needs and if the internal
681 perspective of creating the system with a procedural mindset clashes with the
682 outside perspective (i.e., pillar 2).

683 **(2) The usage of the IWS.** We investigate the external quality attributes of using
684 an IWS from the IWS *consumer’s* perspective. That is, we ask if existing

685 validation techniques are sufficient enough to ensure that the end-users can
 686 actually use an IWS to build their software in the ways they expect the IWS to
 687 work.

688 **(3) The nature of an IWS.** We investigate what standard software engineering
 689 practices apply when developing probabilistic systems. That is, we tackle what
 690 best practices exist when developing systems that are inherently stochastic and
 691 probabilistic, i.e., the ‘black box’ intelligence itself.

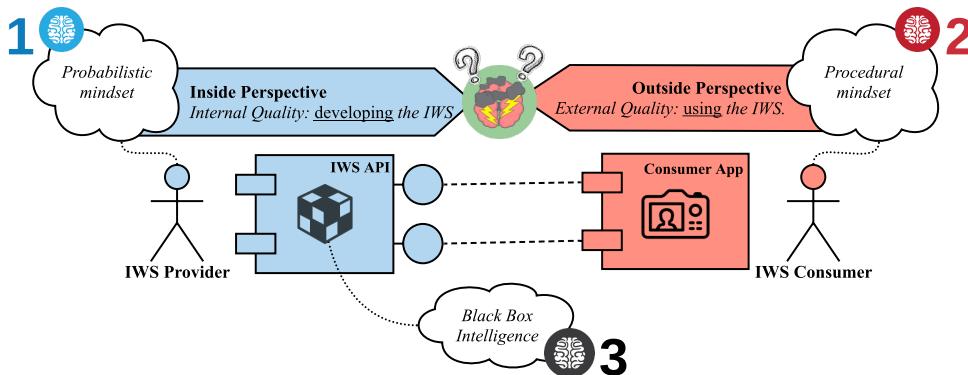


Figure 2.1: The three pillars by which we anchor the background: (1) developing an IWS with a probabilistic mindset by the IWS provider; (2) the use of a IWS with a procedural mindset by the IWS consumer; (3) the nature of a IWS itself.

692 Does a clash of procedural consumer mindsets who use a IWS and the proba-
 693 bilistic provider mindsets who develop them exist? And what impact does this have
 694 on the inside and outside perspective? Throughout this chapter, we will review these
 695 three core pillars due to such mindset mismatch from the anchoring perspective of
 696 software quality, particularly around verification & validation (V&V) and related
 697 quality attributes, probabilistic and non-deterministic software and the nature of
 698 APIs.

699 2.1 Software Quality

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

ROBERT PIRSIG, 1974 [286]

700 The philosophical viewpoint of ‘quality’ remains highly debated and there are mul-
 701 tiple facets to perceive this complex concept [132]. Transcendentally, a viewpoint
 702 like that of Pirsig’s above shows that quality is not tangible but still recognisable; it’s
 703 hard to explicitly define but you know when it’s missing. The International Orga-
 704 nization for Standardization provides a breakdown of seven universally-applicable
 705 principles that defines quality for organisations, developers, customers and training
 706 providers [171]. More pertinently, the 1986 ISO standard for quality was simply
 707 “the totality of characteristics of an entity that bear on its ability to satisfy stated or

708 implied needs” [170].

709 Using this sentence, what characteristics exist for non-deterministic IWSs like
710 that of a CVS? How do we know when the system has satisfied its ‘stated or implied
711 needs’ when the system can only give us uncertain probabilities in its outputs? Such
712 answers can be derived from related definitions—such as ‘conformance to specifica-
713 tion or requirements’ [85, 138], ‘meeting or exceeding customer expectation’ [37],
714 or ‘fitness for use’ [184]—but these then still depend on the solution description or
715 requirements specification, and thus the same questions still apply.

716 *Software* quality is somewhat more concrete. Pressman [288] adapted the
717 manufacturing-oriented view of quality from [38] and phrased software quality
718 under three core pillars:

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

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

739 2.1.1 Validation and Verification

740 To explain V&V, we analogously recount a tale given by Pham [284] on his works
741 on reliability. A high-school student sat a standardised test that was sent to 350,0000
742 students [347]. A multiple-choice algebraic equation problem used a variable, *a*,
743 and intended that students *assume* that the variable was non-negative. Without
744 making this assumption explicit, there were two correct answers to the multiple
745 choice answer. Up to 45,000 students had their scores retrospectively boosted by up
746 to 30 points for those who ‘incorrectly’ answered, however, outcomes of a student’s
747 higher education were, thereby, affected by this one oversight in quality assessment.
748 The examiners wrote a poor question due to poor process standards to check if
749 their ‘correct’ answers were actually correct. The examiners “didn’t build the right

750 product” nor did they “build the product right” by writing a poor question and failing
751 to ensure quality standards, in the phrases Boehm [48] coined.

752 This story describes the issues with the cost of quality [47] and the importance
753 of V&V: just as the poorly written exam question had such a high toll on the 45,000
754 unlucky students, so does poorly written software in production. As summarised by
755 Pressman [288], data sourced from Digital [79] in a large-scale application showed
756 that the difference in cost to fix a bug in development versus system testing is
757 \$6,159 per error. In safety-critical systems, such as self-driving cars or clinical
758 decision support systems, this cost skyrockets due to the extreme discipline needed
759 to minimise error [350].

760 Formally, we refer to the IEEE Standard Glossary of Software Engineering
761 Terminology [167] for to define V&V:

762 **verification** The process of evaluating a system or component to determine
763 whether the products of a given development phase satisfy the
764 conditions imposed at the start of that phase.

765 **validation** The process of evaluating a system or component during or at the
766 end of the development process to determine whether it satisfies
767 specified requirements.

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

770 The verification process of API providers ‘leak’ out to the context of the de-
771 veloper’s project dependent on the IWS. Poor verification in the *internal quality*
772 of the IWS will entail poor process standards, such as poor definitions and termi-
773 nology used, support tooling and description of documentations [335]. Though
774 it is commonplace for providers to have a ‘ship-first-fix-later’ mentality of ‘good-
775 enough’ software [363], the consequence of doing so leads to consumers absorbing
776 the cost. Thus API providers must ensure that their verification strategies
777 are rigorous enough for the consumers in the myriad contexts they wish to use
778 it in. Studies have considered V&V in the context of web services on the cloud
779 [21, 69, 70, 120, 156, 255, 257, 386], though little have recently considered how
780 adding ‘intelligence’ to these services affects existing proposed frameworks and
781 solutions. For a CVS, what might this entail? Which assurances are given to the
782 consumers, and how is that information communicated? To verify if the service is
783 working correctly, does that mean that we need to deploy the system first to get a
784 wider range of data, given the stochastic nature of the black box?

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

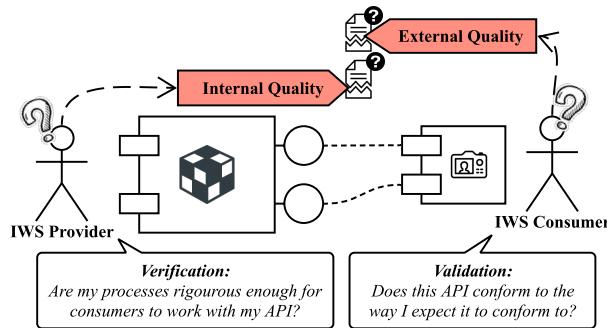


Figure 2.2: The ‘leakage’ of internal quality into the API consumer’s product and external quality imposing on the API provider.

793 Therefore, just as the test answers were both correct and incorrect at the
 794 same time, so is the same with IWSs returning a probabilistic result: no result is
 795 certain. While V&V has been investigated in the area of mathematical and earth
 796 sciences for numerical probabilistic models and natural systems [267, 313], from
 797 the software engineering literature, little work has been achieved to look at the
 798 surrounding area of probabilistic systems hidden behind API calls.

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

811 2.1.2 Quality Attributes and Models

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

816 Quality attribute assessment models (like those shown in Figure 2.3) are an early
 817 concept in software engineering, and systematically evaluating software quality ap-
 818 pears as early as 1968 [312]. Rubey and Hartwick’s 1968 study introduced the phrase
 819 ‘attributes’ as a “prose expression of the particular quality of desired software” (as
 820 worded by Boehm et al. [46]) and ‘metrics’ as mathematical parameters on a scale of

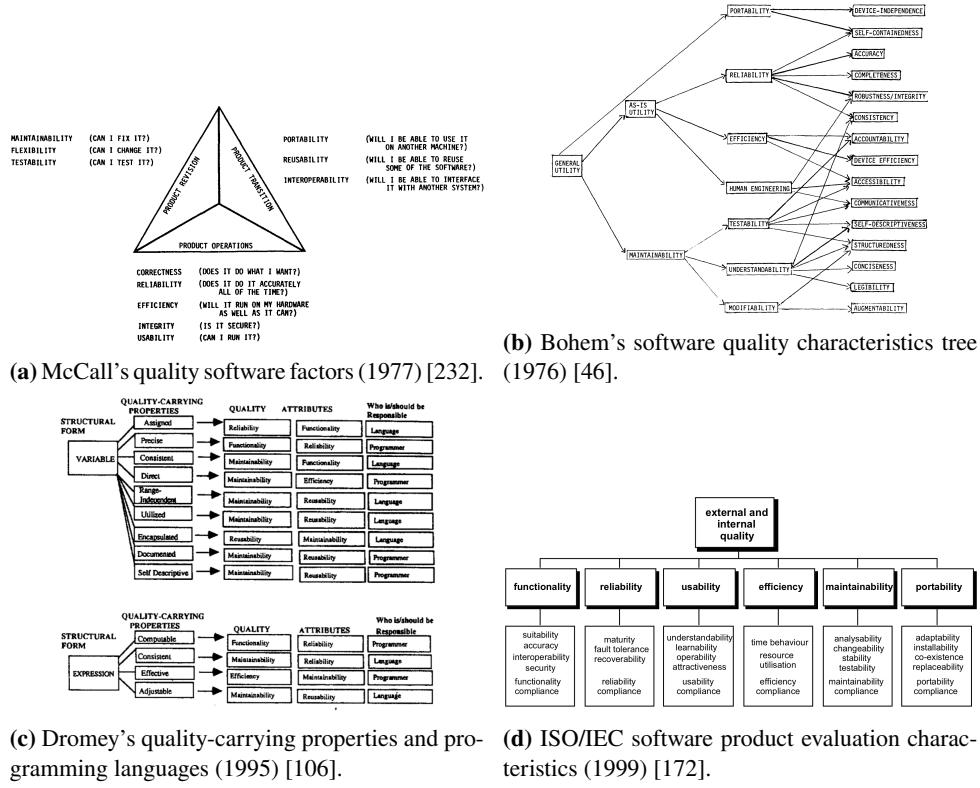


Figure 2.3: A brief overview of the development of software quality models since 1977.

0 to 100. Early attempts to categorise wider factors under a framework was proposed by McCall, Richards, and Walters in the late 1970s [73, 232]. This model described quality from the three perspectives of product revision (*how can we keep the system operational?*), transition (*how can we migrate the system as needed?*) and operation (*how effective is the system at achieving its tasks?*) (Figure 2.3a). The model also introduced 11 attributes alongside numerous direct and indirect measures to help quantify quality. This model was further developed by Boehm et al. [46] who independently developed a similar model, starting with an initial set of 11 software characteristics. It further defined candidate measurements of Fortran code to such characteristics, taking shape in a tree-like structure as in Figure 2.3b. In the mid-1990s, Dromey's interpretation [106] defined a set of quality-carrying properties with structural forms associated to specific programming languages and conventions (Figure 2.3c). The model also supported quality defect identification and proposed an improved auditing method to automate defect detection for code editors in integrated development environments (IDEs). As the need for quality models became prevalent, the International Organization for Standardization standardised software quality under ISO/IEC-9126 [172] (the Software Product Evaluation Characteristics, Figure 2.3d), which has since recently been revised to ISO/IEC-25010 with the introduction of the Systems and software Quality Requirements and Evaluation (SQuaRE) model [169], separating quality into *Product Quality* (consisting of eight

⁸⁴¹ quality characteristics and 31 sub-characteristics) and *Quality In Use* (consisting of
⁸⁴² five quality characteristics and 9 sub-characteristics). An extensive review on the
⁸⁴³ development of quality models in software engineering is given in [6].

⁸⁴⁴ Of all the models described, there is one quality attribute that relates most
⁸⁴⁵ with our narrative of IWS quality: reliability. Reliability is the primary quality
⁸⁴⁶ factor investigated within this thesis (see Section 1.4). Both McCall and Boehm's
⁸⁴⁷ quality models have sub-characteristics of reliability relating to the primary research
⁸⁴⁸ questions that investigate the *robustness*, *consistency* and *completeness*¹ of CVSs
⁸⁴⁹ and its associated documentation. Moreover, the definition of reliability is similar
⁸⁵⁰ among all quality models:

⁸⁵¹ **McCall et al.** Extent to which a program can be expected to perform its in-
⁸⁵² tended function with required precision [232].

⁸⁵³ **Boehm et al.** Code possesses the characteristic *reliability* to the extent that
⁸⁵⁴ it can be expected to perform its intended functions satisfac-
⁸⁵⁵ torily [46].

⁸⁵⁶ **Dromey** Functionality implies reliability. The reliability of software is
⁸⁵⁷ therefore dependent on the same properties as functionality, that
⁸⁵⁸ is, the correctness properties of a program [106].

⁸⁵⁹ **ISO/IEC-9126** The capability of the software product to maintain a specified
⁸⁶⁰ level of performance when used under specified conditions [172].

⁸⁶¹ These definitions strongly relate to the system's solution description in that
⁸⁶² reliability is the ability to maintain its *functionality* under given conditions. But what
⁸⁶³ defines reliability when the nature of an IWS in itself is inherently unpredictable
⁸⁶⁴ due to its probabilistic implementation? Can a non-deterministic system ever be
⁸⁶⁵ considered reliable when the output of the system is uncertain? How do developers
⁸⁶⁶ perceive these quality aspects of reliability in the context of such systems? A system
⁸⁶⁷ cannot be perceived as 'reliable' if the system cannot reproduce the same results due
⁸⁶⁸ to a probabilistic nature. Therefore, we believe the literature of quality models does
⁸⁶⁹ not suffice in the context of IWS reliability; a CVS can interpret an image of a dog
⁸⁷⁰ as a 'Dog' one day, but what if the next it interprets such image more specifically to
⁸⁷¹ the breed, such as 'Border Collie'? Does this now mean the system is unreliable?

⁸⁷² Moreover, defining these systems in themselves is challenging when require-
⁸⁷³ ments specifications and solution descriptions are dependent on nondeterministic
⁸⁷⁴ and probabilistic algorithms. We discuss this further in Section 2.2.

⁸⁷⁵ 2.1.3 Reliability in Computer Vision

⁸⁷⁶ Testing computer vision deep-learning reliability is an area explored typically
⁸⁷⁷ through the use of adversarial examples [345]. These input examples are where

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.

878 images are slightly perturbed to maximise prediction error but are still interpretable
879 to humans. Refer to Figure 2.4.

880 Google Cloud Vision, for instance, fails to correctly classify adversarial examples
881 when noise is added to the original images [162]. Rosenfeld et al. [310] illustrated
882 that inserting synthetic foreign objects to input images (e.g., a cartoon elephant)
883 can alter classification output. Wang et al. [366] performed similar attacks on a
884 transfer-learning approach of facial recognition by modifying pixels of a celebrity’s
885 face to be recognised as a different celebrity, all while still retaining the same human-
886 interpretable original celebrity. Su et al. [340] used the ImageNet database to show
887 that 41.22% of images drop in confidence when just a *single pixel* is changed in the
888 input image; and similarly, Eykholt et al. [113] recently showed similar results that
889 made a CNN interpret a stop road-sign (with mimicked graffiti) as a 45mph speed
890 limit sign.

891 Thus, the state-of-the-art computer vision techniques may not be reliable enough
892 for safety critical applications (such as self-driving cars) as they do not handle inten-
893 tional or unintentional adversarial attacks. Moreover, as such adversarial examples
894 exist in the physical world [113, 206], “the real world may be adversarial enough”
895 [283] to fool such software.

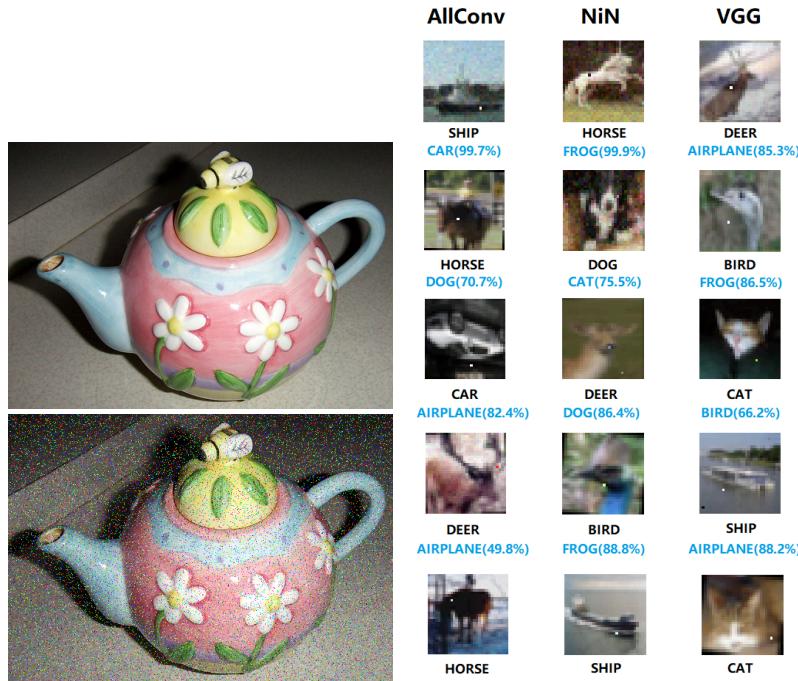
896 2.2 Probabilistic and Nondeterministic Systems

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

902 2.2.1 Interpreting the Uninterpretable

903 As the rise of applied AI increases, the need for engineering interpretability around
904 models becomes paramount, chiefly from an external quality perspective that the
905 *reliability* of the system can be inspected by end-users. Model interpretability has
906 been stressed since early machine learning research in the late 1980s and 1990s (such
907 as Quinlan [290] and Michie [243]), and although there has since been a significant
908 body of work in the area [19, 35, 53, 66, 97, 115, 126, 136, 182, 215, 219, 230, 278,
909 299, 311, 332, 361, 364], it is evident that ‘accuracy’ or model ‘confidence’ is still
910 used as a primary criterion for AI evaluation [165, 175, 334]. Much research into
911 neural network (NN) or support vector machine (SVM) development stresses that
912 ‘good’ models are those with high accuracy. However, is accuracy enough to justify
913 a model’s quality?

914 To answer this, we revisit what it means for a model to be accurate. Accuracy
915 is an indicator for estimating how well a model’s algorithm will work with future
916 or unforeseen data. It is quantified in the AI testing stage, whereby the algorithm
917 is tested against cases known by humans to have ground truth but such cases are
918 unknown by the algorithm. In production, however, all cases are unknown by both



(a) Adding 10% impulse noise to an image of a teapot changes Google Cloud Vision's label from *teapot* (above) to *biology* (below) [162].

(b) One-pixel attacks applied to three neural network (NN): AllConv, NiN and VGG [340].



(c) Adversarial examples to trick face recognition from the source to target images [366].

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

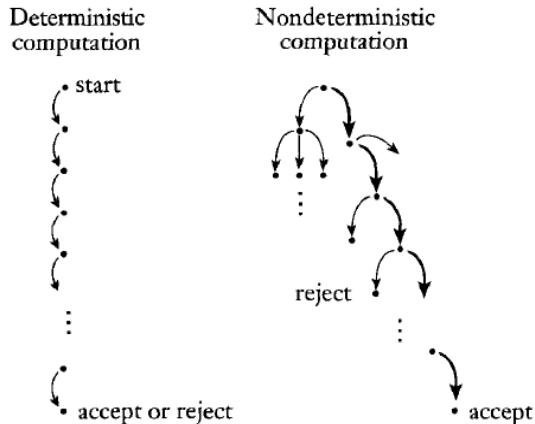


Figure 2.5: A deterministic system (left) always returns the same result in the same amount of steps. A nondeterministic system does not guarantee the same outcome, even with the same input data. Source: [117].

the algorithm *and* the humans behind it, and therefore a single value of quality is “not reliable if the future dataset has a probability distribution significantly different from past data” [122], a problem commonly referred to as the *datashift* problem [317]. Analogously, Freitas [122] provides the following description of the problem:

The military trained [a NN] to classify images of tanks into enemy and friendly tanks. However, when the [NN] was deployed in the field (corresponding to “future data”), it had a poor accuracy rate. Later, users noted that all photos of friendly (enemy) tanks were taken on a sunny (overcast) day. I.e., the [NN] learned to discriminate between the colors of the sky in sunny vs. overcast days! If the [NN] had output a comprehensible model (explaining that it was discriminating between colors at the top of the images), such a trivial mistake would immediately be noted. [122]

So, why must we interpret models? While the formal definition of what it means to be *interpretable* is still somewhat disparate (though some suggestions have been proposed [219]), what is known is (i) there exists a critical trade-off between accuracy and interpretability [102, 121, 145, 181, 189, 388], and (ii) a single quantifiable value cannot satisfy the subjective needs of end-users [122]. As ever-growing domains ML become widespread², these applications engage end-users for real-world goals, unlike the aims in early ML research where the aim was to get AI working in the first place. In safety-critical systems where AI provide informativeness to humans to make the final call (see [71, 166, 192]), there is often a mismatch between the formal objectives of the model (e.g., to minimise error) and complex real-world goals, where other considerations (such as the human factors and cognitive science

²In areas such as medicine [34, 66, 111, 176, 182, 210, 279, 301, 361, 383, 392], bioinformatics [101, 123, 178, 188, 344], finance [19, 99, 166] and customer analytics [215, 364].

943 behind explanations³) are not realised: model optimisation is only worthwhile if they
 944 “actually solve the original [human-centred] task of providing explanation” [256]
 945 to end-users. **Therefore, when human-decision makers must be interpretable**
 946 **themselves [302], any AI they depend on must also be interpretable.**

947 Recently, discussion behind such a notion to provide legal implications of in-
 948 terpretability is topical. Doshi-Velez et al. [105] discuss when explanations are not
 949 provided from a legal stance—for instance, those affected by algorithmic-based de-
 950 cisions have a ‘right to explanation’ [227, 365] under the European Union’s GDPR⁴.
 951 But, explanations are not the only way to ensure AI accountability: theoretical guar-
 952 antees (mathematical proofs) or statistical evidence can also serve as guarantees
 953 [105], however, in terms of explanations, what form they take and how they are
 954 proven correct are still open questions [219].

955 2.2.2 Explanation and Communication

956 From a software engineering perspective, explanations and interpretability are, by
 957 definition, inherently communication issues: what lacks here is a consistent interface
 958 between the AI system and the person using it. The ability to encode ‘common
 959 sense reasoning’ [233] into programs today has been achieved, but *decoding* that
 960 information is what still remains problematic. At a high level, Shannon and Weaver’s
 961 theory of communication [325] applies, just as others have done with similar issues in
 962 the software engineering realm [247, 377] (albeit to the domain of visual notations).
 963 Humans map the world in higher-level concepts easily when compared to AI systems:
 964 while we think of a tree first (not the photons of light or atoms that make up the
 965 tree), an algorithm simply sees pixels, and not the concrete object [105] and the AI
 966 interprets the tree inversely to humans. Therefore, the interpretation or explanation
 967 is done inversely: humans do not explain the individual neurons fired to explain their
 968 predictions, and therefore the algorithmic transparent explanations of AI algorithms
 969 (“*which neurons were fired to make this AI think this tree is a tree?*”) do not work
 970 here.

971 Therefore, to the user (as mapped using Shannon and Weaver’s theory), an AI
 972 pipeline (the communication *channel*) begins with a real-world concept, y , that acts
 973 as an *information source*. This information source is fed in as a *message*, x , (as pixels)
 974 to an AI system (the *transmitter*). The transmitter encodes the pixels to a prediction,
 975 \hat{y} , the *signal* of the message. This signal is decoded by the *receiver*, an explanation
 976 system, $e_x(x, \hat{y})$, that tailors the prediction with the given input data to the intended
 977 end user (the *destination*) as an explanation, \tilde{y} , another type of *message*. Therefore,
 978 the user only sees the channel as an input/output pipeline of real-world objects, y ,
 979 and explanations, \tilde{y} , tailored to *them*, without needing to see the inner-mechanics of
 980 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.

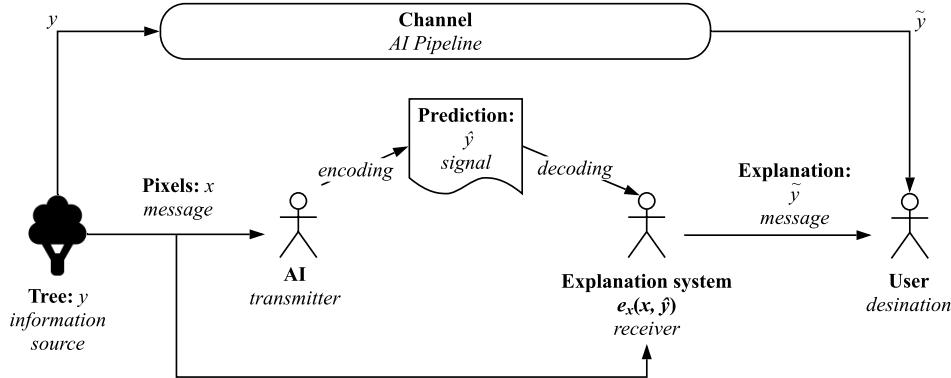


Figure 2.6: Theory of AI communication from information source, y , to intended user as explanations, \tilde{y} .

2.2.3 Mechanics of Model Interpretation

How do we interpret models? Methods for developing interpretation models include: decision trees [59, 83, 153, 224, 291], decision tables [20, 215] and decision sets [208, 256]; input gradients, gradient vectors or sensitivity analysis [19, 212, 299, 311, 322]; exemplars [124, 193]; generalised additive models [71]; classification (*if-then*) rules [55, 80, 270, 354, 380] and falling rule lists [332]; nearest neighbours [230, 294, 323, 375, 389] and Naïve Bayes analysis [34, 75, 114, 125, 157, 200, 210, 392].

Cross-domain studies have assessed the interpretability of these techniques against end-users, measuring response time, accuracy in model response and user confidence [7, 123, 154, 166, 230, 316, 341, 364], although it is generally agreed that decision rules and decision tables provide the most interpretation in non-linear models such as SVMs or NNs [123, 230, 364]. For an extensive survey of the benefits and fallbacks of these techniques, we refer to Freitas [122], Doshi-Velez et al. [105] and Doshi-Velez and Kim [104].

An important factor in model interpretation is to avoid over-reliance, and thus, one mechanism of model interpretation is to reduce explanations altogether. For example, Bussone et al. [66] showed that, in clinical decision support systems, confidence values alone only results in a slight effect on trust and reliance of a system. However, having overly detailed explanations may also cause over-reliance on systems if explanations are detailed but not necessarily true [66]. Hence, a mechanism of model interpretation for the purpose of ensuring trust and reliance is to deliberately show *fewer* explanations or *incorrect* explanations, thereby avoiding over-reliance. A balance between under-explained and overly-explained models is required. This is to encourage intuition in users of a system; similarly, in Ribeiro et al. [299], it was shown that accuracy alone is not always the best way to ascertain trust. Thus, intuitive factors are also mechanisms that can be encoded into explainable models.

1009 2.3 Application Programming Interfaces

1010 Application programming interfaces (APIs) are the interface between a developer
1011 needs and the software components at their disposal [14] by abstracting the underlying
1012 component behind a subroutine, protocol or specific tool. Therefore, it is natural
1013 to assess internal quality (and external quality if the software is in itself a service to
1014 be used by other developers—in this case an IWS) is therefore directly related to the
1015 quality the API offers [199].

1016 Good APIs are known to be intuitive and require less documentation browsing
1017 [285], thereby increasing developer productivity. Conversely, poor APIs are those
1018 that are hard to interpret, thereby reducing developer productivity and product quality.
1019 The consequences of this have shown a higher demand of technical support (as
1020 measured in [158]) that, ultimately, causes the maintenance to be far more expensive,
1021 a phenomenon widely known in software engineering economics (see Section 2.1.1).

1022 While there are different types of APIs, such as software library/framework
1023 APIs for building desktop software, operating system APIs for interacting with the
1024 operating system, remote APIs for communication of varying technologies through
1025 common protocols, we focus on web APIs for communication of resources over
1026 the web (being the common architecture of cloud-based services). Further infor-
1027 mation on the development, usage and documentation of web APIs is provided in
1028 Section A.1.

1029 2.3.1 API Usability

1030 If a developer doesn't understand the overarching concepts of the context behind
1031 the API they wish to use, then they cannot formulate what gaps in their knowledge
1032 is missing. For example, a developer that knows nothing about ML techniques in
1033 computer vision cannot effectively formulate queries to help bridge those gaps in
1034 their understanding to figure out more about the CVS they wish to use.

1035 Balancing the understanding of the information need (both conscious and un-
1036 conscious), how to phrase that need and how to query it in an information retrieval
1037 system is concept long studied in the information sciences [352]. In API design,
1038 the most common form to convey knowledge to developers is through annotated
1039 code examples and overviews to a platform's architectural and design decisions
1040 [56, 103, 253, 305] though these studies have not effectively communicated *why*
1041 these artefacts are important. What makes the developer *conceptually understand*
1042 these artefacts?

1043 Robillard and Deline [305] conducted a multi-phase, mixed-method approach to
1044 create knowledge grounded in the professional experience of 440 software engineers
1045 at Microsoft of varying experience to determine what makes APIs hard to learn,
1046 the results of which previously published in an earlier report [304]. Their results
1047 demonstrate that 'documentation-related obstacles' are the biggest hurdle in learning
1048 new APIs. One of these implications are the *intent documentation* of an API (i.e.,
1049 *what is the intent for using a particular API?*) and such documentation is required
1050 only where correct API usage is not self-evident, where advanced uses of the API are

1051 documented (but not the intent), and where performance aspects of the API impact
1052 the application developed using it. They conclude that professional developers do
1053 not struggle with learning the *mechanics* of the API, but in the *understanding* of how
1054 the API fits in upwards to its problem domain and downward to its implementation:

1055 *In the upwards direction, the study found that developers need help
1056 mapping desired scenarios in the problem domain to the content of the
1057 API, and in understanding what scenarios or usage patterns the API
1058 provider intends and does not intend to support. In the downwards
1059 direction, developers want to understand how the API's implementation
1060 consumes resources, reports errors and has side effects.* [305]

1061 These results particularly corroborate to that of previous studies where devel-
1062 opers quote that they feel that existing learning content currently focuses on “*how*
1063 to do things, not necessarily *why*” [264]. This thereby reiterates the conceptual
1064 understanding of an API as paramount.

1065 A later study by Ko and Riche [198] assessed the importance of a programmer’s
1066 conceptual understanding of the background behind the task before implementing the
1067 task itself, a notion that we find most relevant for users of IWS APIs. While the study
1068 did not focus on developing web APIs (rather implementing a Bluetooth application
1069 using platform-agnostic terminology), the study demonstrated how developers show
1070 little confidence in their own metacognitive judgements to understand and assess the
1071 feasibility of the intent of the API and understand the vocabulary and concepts within
1072 the domain (i.e., wireless connectivity). This indecision over what search results
1073 were relevant in their searches ultimately hindered their progress implementing the
1074 functionality, again decreasing productivity. Ko and Riche suggest to improve API
1075 usability by introducing the background of the API and its relevant concepts using
1076 glossaries linked to tutorials to each of the major concepts, and then relate it back to
1077 how to implement the particular functionality. Thus, an analysis of the conceptual
1078 understanding of IWS APIs by a range of developers (from beginner to professional)
1079 is critical to best understand any differences between existing studies and those that
1080 are nondeterministic.

1081 2.4 Summary

1082 This background chapter explored nuances of interacting and integrating with proba-
1083 bilistic components, namely IWSs, and the impacts this may have to software quality.
1084 Firstly, we explored both internal and external quality attributes of IWSs and how
1085 leakage of internal quality may affect the external quality of client applications.
1086 We discussed how V&V approaches can assist in improving quality assurance of
1087 probabilistic components, and reviewed how various software quality attributes and
1088 models emphasise reliability of systems and their associated documentation (namely,
1089 through the sub-characteristics of robustness, consistency and completeness). We
1090 applied this context to CVSs, giving examples where these cloud services may not
1091 be reliable. Lastly, we applied the narrative of reliability to the overarching nature

¹⁰⁹² of computer vision itself, exploring how the underlying ML models behind a CVS
¹⁰⁹³ can potentially fail, and discussed how any such ML model should be explainable
¹⁰⁹⁴ to ensure its reliability and trustworthiness. Lastly, we discussed the impact an API
¹⁰⁹⁵ can have when it is of poor quality, again impacting the internal quality of a system.
¹⁰⁹⁶ In the next chapter, we propose several research strategies in the search for further
¹⁰⁹⁷ insight into the developer's approach toward existing IWS APIs.

CHAPTER 3

1098

1099

1100

Research Methodology

1101

1102 Investigating software engineering practices is often a complex task as it is imper-
1103 ative to understand the social and cognitive processes around software engineers
1104 and not just the tools and processes used [109]. This chapter explores our research
1105 methodology by exploring five key elements of empirical software engineering re-
1106 search: firstly, (i) we provide an extended focus to the study by reviewing our research
1107 questions (see Section 1.4) anchored under the context of an existing research ques-
1108 tion classification taxonomy, (ii) characterise our research goals through an explicit
1109 philosophical stance, (iii) explain how the stance selected impacts our selection of
1110 research methods and data collection techniques (by dissecting our choice of meth-
1111 ods used to reach these research goals), (iv) discuss a set of criteria for assessing the
1112 validity of our study design and the findings of our research, and lastly (v) discuss
1113 the practical considerations of our chosen methods.

1114 The foundations for developing this research methodology has been expanded
1115 from that proposed by Easterbrook et al. [109], Wohlin and Aurum [381], Wohlin
1116 et al. [382] and Shaw [327].

1117 3.1 Research Questions Revisited

1118 To discuss our research strategy, we revisit our four primary and seven secondary
1119 research questions (RQs) through the classification technique discussed by Easter-
1120 brook et al. [109], a technique originally proposed in the field of psychology by
1121 Meltzoff and Cooper [238] but adapted to software engineering. A summary of the
1122 classifications made to our research questions are presented in Table 3.1.

1123 Our research study involves a mix of nine *empirical*¹ RQs, that focus on observ-
1124 ing and analysing existing phenomena, and two *non-empirical* RQs, that focuses
1125 on designing better approaches to solve software engineering tasks [242]. The use

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

¹¹²⁶ of empirical *and* non-empirical RQs are best combined in long-term software en-
¹¹²⁷ gineering research studies where the phenomena are under-explored, as is the case
¹¹²⁸ with CVSs. Further, these approaches help propose solutions to issues found in the
¹¹²⁹ phenomena studied [378]. We discuss both our empirical and non-empirical RQs in
¹¹³⁰ 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. [109] and Meltzoff and Cooper [238].

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

¹¹³¹ 3.1.1 Empirical Research Questions

¹¹³² In total, we pose nine empirically-based RQs to help us understand the way develop-
¹¹³³ ers currently interact and work with web services that provide computer vision. The
¹¹³⁴ majority of these questions are *exploratory* questions that contribute to a landscape
¹¹³⁵ analysis of these services (RQ1, RQ1.1 and RQ1.2), how well they are documented
¹¹³⁶ (RQ2), and the issues developers currently face when using them (RQ3). Our other

1137 exploratory questions complement the answers to these questions. For instance, to
1138 understand if CVSs are sufficiently documented (an *existence* exploratory question
1139 posed in RQ2), we need to understand the components of a ‘sufficient’ or ‘com-
1140 plete’ API document via RQ2.1 as proposed in both the literature and by software
1141 developers. While RQ2.1 does not directly relate to CVSs, answering it gives us
1142 an understanding the components of complete API documentation, and therefore,
1143 we can assess what aspects they are missing and where improvements can be made
1144 (RQ2.2). These questions are *descriptive and classification* questions that help de-
1145 scribe and classify what practices are in use for existing CVS API documentation
1146 and the nature behind these services. Answering these exploratory questions assists
1147 in refining preciser terms of the phenomena, ways in which we find evidence for
1148 them, and ensuring the data found is valid.

1149 By answering these questions, we have a clearer understanding of the phenom-
1150 ena; we then follow up by posing two additional *base-rate questions* that helps
1151 provide a basis to confirm that the phenomena occurring is normal (or unusual)
1152 behaviour by investigating the patterns of phenomena’s occurrence against other
1153 phenomena. RQ3.1 is a *frequency and distribution* question to help us understand
1154 what types of issues developers often encounter most, given a lack of formal extended
1155 training in artificial intelligence. This achieves us an insight into the developer’s
1156 mindset and regular thought patterns toward these APIs. We can then contrast
1157 this distribution using our second base-rate question (RQ3.3), that assesses the
1158 distributional differences between these intelligent components and non-intelligent
1159 (conventional) software components. Combined, these two questions can help us
1160 answer how the issues raised against CVSs are different to normal Stack Overflow
1161 issues—our *descriptive-comparative* question posed in RQ3—and, similarly, we can
1162 classify and rank which issues developers find most frustrating (RQ3.2).

1163 3.1.2 Non-Empirical Research Questions

1164 RQ2.2 and RQ4 are both non-empirically-based *design questions*; they are con-
1165 cerned with ways in which we can improve a CVS by investigating what additional
1166 attributes are needed in both the documentation of CVSs and in the integration
1167 architectures developers can employ to improve reliability and robustness in their
1168 applications. They are not classified as empirical questions as we investigate what
1169 *will be* and not *what is*. By understanding the process by which developers desire
1170 additional attributes of documentation and integration strategies, we can help shape
1171 improvements to the existing designs of using CVSs.

1172 3.2 Philosophical Stances

1173 Philosophical stances guide the researcher’s action by fortifying what constitutes
1174 ‘valid truth’ against a fundamental set of core beliefs [303]. In software engineer-
1175 ing, four dominant philosophical stances are commonly characterised [84, 281]:
1176 positivism (or post-positivism), constructivism (or interpretivism), pragmatism, and
1177 critical theory (or advocacy/participatory). To construct such a ‘validity of truth’,

¹¹⁷⁸ we will review these four philosophical stances in this section, and state the stance
¹¹⁷⁹ that we explicitly adopt and our reasoning for this.

¹¹⁸⁰ *Positivism*

¹¹⁸¹ Positivists claim truth to be all observable facts, reduced piece-by-piece to smaller
¹¹⁸² components which is incrementally verifiable to form truth. We do not base our
¹¹⁸³ work on the positivistic stance as the theories governing verifiable hypothesis must
¹¹⁸⁴ be precise from the start of the research. Moreover, due to its reductionist approach,
¹¹⁸⁵ it is difficult to isolate these hypotheses and study them in isolation from context.
¹¹⁸⁶ As our hypotheses are not context-agnostic, we steer clear from this stance.

¹¹⁸⁷ *Constructivism*

¹¹⁸⁸ Constructivists see knowledge embedded within the human context; truth is the
¹¹⁸⁹ *interpretive* observation by understanding the differences in human thought between
¹¹⁹⁰ meaning and action [197]. That is, the interpretation of the theory is just as important
¹¹⁹¹ to the empirical observation itself. We partially adopt a constructivist stance as we
¹¹⁹² attempt to model the developer's mindset, being an approach that is rich in qualitative
¹¹⁹³ data on human activity.

¹¹⁹⁴ *Pragmatism*

¹¹⁹⁵ Pragmatism is a less dogmatic approach that encourages the incomplete and approx-
¹¹⁹⁶ imate nature of knowledge and is dependent on the methods in which the knowledge
¹¹⁹⁷ was extracted. The utility of consensually agreed knowledge is the key outcome, and
¹¹⁹⁸ is therefore relative to those who seek utility in the knowledge—what is the useful
¹¹⁹⁹ for one person is not so for the other. While we value the utility of knowledge, it is
¹²⁰⁰ difficult to obtain consensus especially on an ill-researched topic such as ours, and
¹²⁰¹ therefore we do not adopt this stance.

¹²⁰² *Critical Theory*

¹²⁰³ This study chiefly adopts the philosophy of critical theory [11]. A key outcome of
¹²⁰⁴ the study is to shift the developer's restrictive deterministic mindset and shed light
¹²⁰⁵ on developing a new framework actively with the developer community that seeks
¹²⁰⁶ to improve the process of using such APIs. In software engineering, critical theory
¹²⁰⁷ is used to “actively [seek] to challenge existing perceptions about software practice”
¹²⁰⁸ [109], and this study utilises such an approach to shift the mindset of CVS consumers
¹²⁰⁹ and providers alike on how the documentation and metadata should not be written
¹²¹⁰ with the ‘traditional’ deterministic mindset at heart. Thus, our key philosophical
¹²¹¹ approach is critical theory to seek out *what-can-be* using partial constructivism to
¹²¹² model the current *what-is*.

3.3 Research Methods

1214 Research methods are “a set of organising principles around which empirical data is
1215 collection and analysed” [109]. Creswell [84] suggests that strong research design
1216 is reflected when the weaknesses of multiple methods complement each other. Us-
1217 ing a mixed-methods approach is therefore commonplace in software engineering
1218 research, typically due to the human-oriented nature investigating how software en-
1219 gineers work both individually (where methods from psychology may be employed)
1220 and together (where methods from sociology may be employed).

1221 Therefore, studies in software engineering are typically performed as field studies
1222 where researchers and developers (or the artefacts they produce) are analysed either
1223 directly or indirectly [331]. The mixed-methods approach combines five classes
1224 of field study methods (or empirical strategies/studies) most relevant in empirical
1225 software engineering research [109, 186, 382]: controlled experiments, case studies,
1226 survey research, ethnographies, and action research. We chiefly adopt a mixed-
1227 methods approach to our work using the *concurrent triangulation* mixed-methods
1228 strategy [231] as it best compensates for weaknesses that exist in all research methods,
1229 and employs the best strengths of others [84].

3.3.1 Review of Relevant Research Methods

1231 Below we review some of the research methods most relevant to our research ques-
1232 tions as refined in Section 3.1 as presented by Easterbrook et al. [109].

3.3.1.1 Controlled Experiments

1233 A controlled experiment is an investigation of a clear, testable hypothesis that guides
1234 the researcher to decide and precisely measure how at least one independent variable
1235 can be manipulated and effect at least one other dependent variable. They determine
1236 if the two variables are related and if a cause-effect relationship exists between
1237 them. The combination of independent variable values is a *treatment*. It is common
1238 to recruit human subjects to perform a task and measure the effect of a randomly
1239 assigned treatment on the subjects, though it is not always possible to achieve
1240 full randomisation in real-life software engineering contexts, in which case a *quasi-*
1241 *experiment* may be employed where subjects are not randomly assigned to treatments.

1242 While we have well-defined RQs, refining them into precise, *measurable* vari-
1243 ables is challenging due to the qualitative nature they present. A well-defined
1244 population is also critical and must be easily accessible; the varied range of begin-
1245 ner to expert software engineers with varied understanding of artificial intelligence
1246 concepts is required to perform controlled experiments, and thus recruitment may
1247 prove challenging. Lastly, the controlled experiment is essentially reductionist by
1248 affecting a small amount of variables of interest and controlling all others. This
1249 approach is too clinical for the practical outcomes by which our research goals aim
1250 for, and is therefore closely tied to the positivist stance.

1252 3.3.1.2 *Case Studies*

1253 Case studies investigate phenomena in their real-life context and are well-suited
1254 when the boundary between context and phenomena is unknown [387]. They offer
1255 understanding of how and why certain phenomena occur, thereby investigating ways
1256 cause-effect relationships can occur. They can be used to test existing theories
1257 (*confirmatory case studies*) by refuting theories in real-world contexts instead of
1258 under laboratory conditions or to generate new hypotheses and build theories during
1259 the initial investigation of some phenomena (*exploratory case studies*).

1260 Case studies are well-suited where the context of a situation plays a role in
1261 the phenomenon being studied. They also lend themselves to purposive sampling
1262 rather than random sampling, and thus it is possible to selectively choose cases that
1263 benefit our research goals and (using our critical theorist stance) select cases that
1264 will actively benefit our participant software engineering audience most to draw
1265 attention to situations regarded as problematic in CVS.

1266 3.3.1.3 *Survey Research*

1267 Survey research identifies characteristics of a broad population of individuals through
1268 direct data collection techniques such as interviews and questionnaires or indepen-
1269 dent techniques such as data logging. Defining that well-defined population is
1270 critical, and selecting a representative sample from it to generalise the data gathered
1271 usually assists in answering base-rate questions.

1272 By identifying representative sample of the population, from beginner to ex-
1273 perienced developers with varying understanding of CVS APIs, we can use survey
1274 research to assist in answering our exploratory and base-rate RQs (see Section 3.1.1)
1275 in determining the qualitative aspects of how individual developers perceive and
1276 work with the existing APIs, either by directly asking them, or by mining third-party
1277 discussion websites such as Stack Overflow (SO). Similarly, we can use this strategy
1278 to assess the developer’s understanding on what makes API documentation sufficient
1279 by assessing whether specific factors suggested from literature are useful according
1280 to developers. However, with direct survey research techniques, low response rates
1281 may prove challenging, especially if no inducements can be offered for participation.

1282 3.3.1.4 *Ethnographies*

1283 Ethnographies investigates the understanding of social interaction within community
1284 through field observation [307]. Resulting ethnographies help understand how soft-
1285 ware engineering technical communities build practices, communication strategies
1286 and perform technical work collaboratively.

1287 Ethnographies require the researcher to be highly trained in observational and
1288 qualitative data analysis, especially if the form of ethnography is participant observa-
1289 tion, whereby the researcher is embedded of the technical community for observation.
1290 This may require the longevity of the study to be far greater than a couple of weeks,
1291 and the researcher must remain part of the project for its duration to develop enough
1292 local theories about how the community functions. While it assists in revealing

1293 subtle but important aspects of work practices within software teams, this study
1294 does not focus on the study of teams, and is therefore not a research method relevant
1295 to this project.

1296 **3.3.1.5 Action Research**

1297 Action researchers simultaneously solve real-world problems while studying the
1298 experience of solving the problem [95] by actively seeking to intervene in the
1299 situation for the purpose of improving it. A precondition is to engage with a
1300 *problem owner* who is willing to collaborate in identifying and solving the problem
1301 faced. The problem must be authentic (a problem worth solving) and must have
1302 new knowledge outcomes for those involved. It is also characterised as an iterative
1303 approach to problem solving, where the knowledge gained from solving the problem
1304 has a desirable solution that empowers the problem owner and researcher.

1305 This research is most associated to our adopted philosophical stance of critical
1306 theory. As this project is being conducted under the Applied Artificial Intelligence
1307 Institute (A^2I^2) collaboratively with engaged industry clients, we have identified a
1308 need for solving an authentic problem that industry faces. The desired outcome
1309 of this project is to facilitate wider change in the usage and development of CVSs;
1310 thus, engaging action research as a potential method throughout the mixed-methods
1311 approach is used in this research.

1312 **3.3.2 Review of Data Collection Techniques for Field Studies**

1313 Singer et al. developed a taxonomy [213, 331] showcasing data collection techniques
1314 in field studies that are used in conjunction with a variety of methods based on the
1315 level of interaction between researcher and software engineer, if any. This taxonomy
1316 is reproduced in Table 3.2, where techniques used in this research study are starred.

1317 **3.4 Research Design**

1318 This section discusses an overview of the design of methods used within the experi-
1319 ments conducted under this thesis. For each experiment, we describe an overview of
1320 the experiment grounded known methods and techniques (Sections 3.3.1 and 3.3.2)
1321 and our approach to analysing the data, as well as relating the selecting method back
1322 to a specific RQ. Details of each experiment presented in this thesis, the coherency
1323 between them, and where they can be found are given in Sections 1.6 and 1.7.

1324 **3.4.1 Landscape Analysis of Computer Vision Services**

1325 To understand the behavioural and evolutionary profiles of CVSs (i.e., RQ1), we em-
1326 ployed a longitudinal study based around a dynamic system analysis [331]. Specif-
1327 ically, we used structured observations of three services using the same dataset to
1328 understand how the responses from these services change with time. Lastly, we

Table 3.2: Questions asked by software engineering researchers (column 2) that can be answered by field study techniques. (Adapted from [331].) Methods used within this research study are starred.

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 modelling	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.

1329 utilised documentation analysis to assess the overall ‘picture’ of how these services are documented. Further details on this experiment is given in **Chapter 4, Section 4.4.**

1332 3.4.2 Utility of API Documentation in Computer Vision Services

1333 To assess whether these services are sufficiently documented (i.e., RQ2), we conducted a systematic mapping study [194, 282] of the various academic sources detailing API documentation knowledge. We then consolidated this information into a structured taxonomy following a systematic taxonomy development method specific to software engineering studies [360].

1338 We then followed the triangulation approach proposed by Mayring [231] to validate the taxonomy by use of a personal opinion survey. Kitchenham and Pfleeger [195] provide an introduction on methods used to conduct personal opinion surveys which we adopted as an initial reference in (i) shaping our survey objectives around our research goals, (ii) designing a cross-sectional survey, (iii) developing and evaluating our survey instrument, (iv) evaluating our instruments, (v) obtaining the data and (vi) analysing the data. We adapted Brooke’s systematic usability scale [61] technique by basing our research questions against a known surveying instrument.

1346 As is good practice in developing questionnaire instruments to evaluate their reliability and validity [220], we evaluated our instrument design by asking colleagues to critique it via pilot studies within A²I². This assisted in identifying any problems with the questionnaire itself and with any issues that may have occurred with the response rate and follow-up procedures.

1351 Findings from the pilot study helped inform us for a widely distributed questionnaire using snow-balling sampling. Ethics approval from the Faculty of Science, Engineering and Built Environment Human Ethics Advisory Group (SEBE HEAG) was approved to externally conduct this survey research (see Chapter E). Further details on these methods are detailed within **Chapter 8, Section 8.3.**

1356 3.4.3 Developer Issues concerning Computer Vision Services

1357 Developers typically congregate in search of discourses on issues they face in online forums, such as Stack Overflow (SO) and Quora, as well as writing their experiences in personal blogs such as Medium. The simplest of these platforms is SO (a sub-community of the Stack Exchange family of targeted communities) that specifically targets developer issues on using a simple Q&A interface, where developers can discuss technical aspects and general software development topics. Moreover, SO is often acknowledged as *the ‘go-to’ place* for developers to find high-quality code snippets that assist in their problems [342].

1365 Thus, to begin understanding the issues developers face when using CVSs and whether there is a substantial difference to conventional domains (i.e., RQ3), we used repository mining on SO to help answer RQ3. Specifically, we selected SO due to its targeted community of developers² and the availability of its publicly

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

¹³⁶⁹ available dataset released as ‘data dumps’ on the Stack Exchange Data Explorer³
¹³⁷⁰ and Google BigQuery⁴. Studies conducted have also used SO to mine developer
¹³⁷¹ discourse [8, 22, 28, 77, 218, 261, 271, 296, 308, 333, 348, 368]. Further details on
¹³⁷² how we approached the design for this study can be found in **Chapter 5, Section 5.4,**
¹³⁷³ **Chapter 6, Section 6.3, and Chapter 7, Section 7.3**

¹³⁷⁴ **3.4.4 Designing Improved Integration Strategies**

¹³⁷⁵ Our improved integration strategies (i.e., RQ4) evolved organically over the duration
¹³⁷⁶ of this research through the use of industry case studies and action research. We
¹³⁷⁷ developed several iterative prototypes to the integration strategies and used a mix
¹³⁷⁸ of statistical and technical evaluations to analyse whether our improved integration
¹³⁷⁹ strategies can prove useful. Further details about these approaches are detailed in
¹³⁸⁰ **Chapter 9, Section 9.5.1 and Chapter 10, Section 10.3 and Chapter 11, Sec-**
¹³⁸¹ **tion 11.5.**

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

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

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

1382

Part II

1383

Publications

CHAPTER 4

1384

1385

1386

Identifying Evolution in Computer Vision Services[†]

1387

1388 **Abstract** Recent advances in artificial intelligence (AI) and machine learning (ML), such
1389 as computer vision, are now available as intelligent web services (IWSs) and their acces-
1390 sibility and simplicity is compelling. Multiple vendors now offer this technology as cloud
1391 services and developers want to leverage these advances to provide value to end-users. How-
1392 ever, there is no firm investigation into the maintenance and evolution risks arising from use
1393 of these IWSs; in particular, their behavioural consistency and transparency of their function-
1394 ality. We evaluated the responses of three different IWSs (specifically computer vision) over
1395 11 months using 3 different data sets, verifying responses against the respective documenta-
1396 tion and assessing evolution risk. We found that there are: (1) inconsistencies in how these
1397 services behave; (2) evolution risk in the responses; and (3) a lack of clear communication
1398 that documents these risks and inconsistencies. We propose a set of recommendations to
1399 both developers and IWS providers to inform risk and assist maintainability.

4.1 Introduction

1400 The availability of intelligent web services (IWSs) has made artificial intelligence
1401 (AI) tooling accessible to software developers and promises a lower entry barrier for
1402 their utilisation. Consider state-of-the-art computer vision analysers, which require
1403 either manually training a deep-learning classifier, or selecting a pre-trained model
1404 and deploying these into an appropriate infrastructure. Either are laborious in time,
1405 and require non-trivial expertise along with a large data set when training or customi-
1406 sation is needed. In contrast, IWSs providing computer vision (i.e., computer vision
1407 services or CVSs such as [397, 409, 410, 411, 418, 422, 430, 431, 432, 436, 450,

[†]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.

451, 484, 485]) abstract these complexities behind a web application programming
452 interface (API) call. This removes the need to understand the complexities required
453 of machine learning (ML), and requires little more than the knowledge on how to
454 use RESTful endpoints. The ubiquity of these services is exemplified through their
455 rapid uptake in applications such as aiding the vision-impaired [94, 298].

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

462 ML typically involves use of statistical techniques that yield components with
463 a non-deterministic external behaviour; that is, for the same given input, different
464 outcomes may result. However, developers, in general, are used to libraries and small
465 components behaving predictably, while systems that rely on ML techniques work
466 on confidence intervals¹ and probabilities. For example, the developer’s mindset
467 suggests that an image of a border collie—if sent to three intelligent computer vision
468 services (CVSs)—would return the label ‘dog’ consistently with time regardless
469 of which service is used. However, one service may yield the specific dog breed,
470 ‘border collie’, another service may yield a permutation of that breed, ‘collie’, and
471 another may yield broader results, such as ‘animal’; each with results of varying
472 confidence values.² Furthermore, the third service may evolve with time, and
473 thus learn that the ‘animal’ is actually a ‘dog’ or even a ‘collie’. The outcomes
474 are thus behaviourally inconsistent between services providing conceptually similar
475 functionality. As a thought exercise, consider if the sub-string function were created
476 using ML techniques—it would perform its operation with a confidence where the
477 expected outcome and the AI inferred output match as a *probability*, rather than a
478 deterministic (constant) outcome. How would this affect the developers’ approach
479 to using such a function? Would they actively take into consideration the non-
480 deterministic nature of the result?

481 Myriad software quality models and software engineering practices advocate
482 maintainability and reliability as primary characteristics; stability, testability, fault
483 tolerance, changeability and maturity are all concerns for quality in software com-
484 ponents [161, 288, 335] and one must factor these in with consideration to soft-
485 ware evolution challenges [141, 142, 240, 241, 353]. However, the effect this
486 non-deterministic behaviour has on quality when masked behind an IWS is still
487 under-explored to date in software engineering literature, to our knowledge. Where
488 software depends on IWSs to achieve functionality, these quality characteristics may
489 not be achieved, and developers need to be wary of the unintended side effects and
490 inconsistency that exists when using non-deterministic components. A CVS may
491 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.

1450 sis, but developers are more likely to approach IWSs with a mindset that anticipates
1451 consistency. Although the documentation does hint at this non-deterministic be-
1452 haviour (i.e., the descriptions of ‘confidence’ in various CVSs suggest they are
1453 not always confident, and thus not deterministic [395, 420, 437]), the integration
1454 mechanisms offered by popular vendors do not seem to fully expose the nuances,
1455 and developers are not yet familiar with the trade-offs.

1456 Do popular CVSs, as they currently stand, offer consistent behaviour, and if not,
1457 how is this conveyed to developers (if it is at all)? If CVSs are to be used in production
1458 services, do they ensure quality under rigorous service quality assurance (SQA)
1459 frameworks [161]? What evolution risk [141, 142, 240, 241] do they pose if these
1460 services change? To our knowledge, few studies have been conducted to investigate
1461 these claims. This paper assesses the consistency, evolution risk and consequent
1462 maintenance issues that may arise when developers use IWSs. We introduce a
1463 motivating example in Section 4.2, discussing related work and our methodology
1464 in Sections 4.3 and 4.4. We present and interpret our findings in Section 4.5. We
1465 argue with quantified evidence that these IWSs can only be considered with a mature
1466 appreciation of risks, and we make a set of recommendations in Section 4.6.

1467 4.2 Motivating Example

1468 Consider Rosa, a software developer, who wants to develop a social media photo-
1469 sharing mobile app that analyses her and her friends photos on Android and iOS.
1470 Rosa wants the app to categorise photos into scenes (e.g., day vs. night, outdoors
1471 vs. indoors), generate brief descriptions of each photo, and catalogue photos of her
1472 friends as well as common objects (e.g., all photos with a dog, all photos on the
1473 beach).

1474 Rather than building a computer vision engine from scratch, Rosa thinks she
1475 can achieve this using one of the popular CVSs (e.g., [397, 409, 410, 411, 418, 422,
1476 430, 431, 432, 436, 450, 451, 484, 485]). However, Rosa comes from a typical
1477 software engineering background with limited knowledge of the underlying deep-
1478 learning techniques and implementations as currently used in computer vision. Not
1479 unexpectedly, she internalises a mindset of how such services work and behave based
1480 on her experience of using software libraries offered by various SDKs. This mindset
1481 assumes that different cloud vendor image processing APIs more-or-less provide
1482 similar functionality, with only minor variations. For example, cloud object storage
1483 for Amazon S3 is both conceptually and behaviourally very similar to that of Google
1484 Cloud Storage or Azure Storage. Rosa assumes the CVSs of these platforms will,
1485 therefore, likely be very similar. Similarly, consider the string libraries Rosa will
1486 use for the app. The conceptual and behavioural similarities are consistent; a string
1487 library in Java (Android) is conceptually very similar to the string library she will
1488 use in Swift (iOS), and likewise both behave similarly by providing the same results
1489 for their respective sub-string functionality. However, **unlike the cloud storage and**
1490 **string libraries, different CVSs often present conceptually similar functionality**
1491 **but are behaviourally very different.** IWS vendors also hide the depth of knowledge
1492 needed to use these effectively—for instance, the training data set and ontologies

1493 used to create these services are hidden in the documentation. Thus, Rosa isn't even
1494 exposed to this knowledge as she reads through the documentation of the providers
1495 and, thus, Rosa makes the following assumptions:

- 1496 • **"I think the responses will be consistent amongst these CVSSs."** When Rosa
1497 uploads a photo of a dog, she would expect them all to respond with 'dog'. If
1498 Rosa decides to switch which service she is using, she expects the ontologies
1499 to be compatible (all CVSSs *surely* return dog for the same image) and therefore
1500 she can expect to plug-in a different service should she feel like it making only
1501 minor code modifications such as which endpoints she is relying on.
- 1502 • **"I think the responses will be constant with time."** When Rosa uploads the
1503 photo of a dog for testing, she expects the response to be the same in 10 weeks
1504 time once her app is in production. Hence, in 10 weeks, the same photo of the
1505 dog should return the same label.

1506 4.3 Related Work

1507 If we were to view CVSSs through the lenses of an SQA framework, robustness,
1508 consistency, and maintainability often feature as quality attributes in myriad soft-
1509 ware quality models (e.g., [172]). Software quality is determined from two key
1510 dimensions: (1) in the evaluation of the end-product (external quality) and (2) the
1511 assurances in the development processes (internal quality) [288]. We discuss both
1512 perspectives of quality within the context of our work in this section.

1513 4.3.1 External Quality

1514 4.3.1.1 Robustness for safety-critical applications

1515 A typical focus of recent work has been to investigate the robustness of deep-
1516 learning within computer vision technique implementation, thereby informing the
1517 effectiveness in the context of the end-product. The common method for this has
1518 been via the use of adversarial examples [345], where input images are slightly
1519 perturbed to maximise prediction error but are still interpretable to humans.

1520 Google Cloud Vision, for instance, fails to correctly classify adversarial examples
1521 when noise is added to the original images [162]. Rosenfeld et al. [310] illustrated
1522 that inserting synthetic foreign objects to input images (e.g., a cartoon elephant)
1523 can completely alter classification output. Wang et al. [366] performed similar
1524 attacks on a transfer-learning approach of facial recognition by modifying pixels of
1525 a celebrity's face to be recognised as a completely different celebrity, all while still
1526 retaining the same human-interpretable original celebrity. Su et al. [340] used the
1527 ImageNet database to show that 41.22% of images drop in confidence when just a
1528 *single pixel* is changed in the input image; and similarly, Eykholt et al. [113] recently
1529 showed similar results that made a convolutional neural network (CNN) interpret a
1530 stop road-sign (with mimicked graffiti) as a 45mph speed limit sign.

1531 The results suggest that current state-of-the-art computer vision techniques may
1532 not be robust enough for safety critical applications as they do not handle intentional

1533 or unintentional adversarial attacks. Moreover, as such adversarial examples exist in
1534 the physical world [113, 206], “the natural world may be adversarial enough” [283]
1535 to fool AI software. Though some limitations and guidelines have been explored
1536 in this area, the perspective of *Intelligent Web Services* is yet to be considered and
1537 specific guidelines do not yet exist when using CVSSs.

1538 **4.3.1.2 Testing strategies in ML applications**

1539 Although much work applies ML techniques to automate testing strategies, there is
1540 only a growing emphasis that considers this in the opposite sense; that is, testing
1541 to ensure the ML product works correctly. There are few reliable test oracles
1542 that ensure if an ML has been implemented to serve its algorithm and use case
1543 purposefully; indeed, “the non-deterministic nature of many training algorithms
1544 makes testing of models even more challenging” [16]. Murphy et al. [250] proposed
1545 a software engineering-based testing approach on ML ranking algorithms to evaluate
1546 the ‘correctness’ of the implementation on a real-world data set and problem domain,
1547 whereby discrepancies were found from the formal mathematical proofs of the ML
1548 algorithm and the implementation.

1549 Recently, Braiek and Khomh [54] conducted a comprehensive review of testing
1550 strategies in ML software, proposing several research directions and recommendations
1551 in how best to apply software engineering testing practices in ML programs.
1552 However, much of the area of this work specifically targets ML engineers, and not
1553 application developers. Little has been investigated on how application developers
1554 perceive and understand ML concepts, given a lack of formal training; we note that
1555 other testing strategies and frameworks proposed (e.g., [58, 249, 260]) are targeted
1556 chiefly to the ML engineer, and not the application developer.

1557 However, Arpteg et al. [16] recently demonstrated (using real-world ML projects)
1558 the developmental challenges posed to developers, particularly those that arise when
1559 there is a lack of transparency on the models used and how to troubleshoot ML
1560 frameworks using traditional software engineering debugging tools. This said, there
1561 is no further investigations into challenges when using the higher, ‘ML friendly’
1562 layers (e.g., IWSs) of the ‘machine learning spectrum’ [268], rather than the ‘lower
1563 layers’ consisting of existing ML frameworks and algorithms targeted toward the
1564 ML community.

1565 **4.3.2 Internal Quality**

1566 **4.3.2.1 Quality metrics for cloud services**

1567 CVSSs are based on cloud computing fundamentals under a subset of the Platform as
1568 a Service (PaaS) model. There has been work in the evaluation of PaaS in terms of
1569 quality attributes [129]: these attributes are exposed using service-level agreements
1570 (SLAs) between vendors and customers, and customers denote their demanded
1571 quality of service (QoS) to ensure the cloud services adhere to measurable KPI
1572 attributes.

1573 Although, popular services, such as cloud object storage, come with strong QoS
1574 agreement, to date IWSs do not come with deep assurances around their performance
1575 and responses, but do offer uptime guarantees. For example, how can Rosa demand
1576 a QoS that ensures all photos of dogs uploaded to her app guarantee the specific dog
1577 breeds are returned so that users can look up their other friend’s ‘border collie’s?
1578 If dog breeds are returned, what ontologies exist for breeds? Are they consistent
1579 with each other, or shortened? (‘Collie’ versus ‘border collie’; ‘staffy’ versus
1580 ‘staffordshire bull terrier’?) For some applications, these unstated QoS metrics
1581 specific to the ML service may have significant legal ramifications.

1582 4.3.2.2 *Web service documentation and documenting ML*

1583 From the *developer’s* perspective, little has been achieved to assess IWS quality
1584 or assure quality of these CVSs. Web services and their APIs are the bridge be-
1585 tween developers’ needs and the software components [14]; therefore, assessing
1586 such CVSs from the quality of their APIs is thereby directly related to the develop-
1587 ment quality [199]. Good APIs should be intuitive and require less documentation
1588 browsing [285], thereby increasing productivity. Conversely, poor APIs that are
1589 hard to understand and work with reduce developer productivity, thereby reducing
1590 product quality. This typically leads to developers congregating on forums such as
1591 Stack Overflow, leading to a repository of unstructured knowledge likely to concern
1592 API design [370]. The consequences of addressing these concerns in development
1593 leads to a higher demand in technical support (as measured in [158]) that, ultimately,
1594 causes the maintenance to be far more expensive, a phenomenon widely known in
1595 software engineering economics [48]. Rosa, for instance, isn’t aware of technical ML
1596 concepts; if she cannot reason about what search results are relevant when brows-
1597 ing the service and understanding functionality, her productivity is significantly
1598 decreased. Conceptual understanding is critical for using APIs, as demonstrated by
1599 Ko and Riche, and the effects of maintenance this may have in the future of her
1600 application is unknown.

1601 Recent attempts to document attributes and characteristics on ML models have
1602 been proposed. Model cards were introduced by Mitchell et al. [245] to describe how
1603 particular models were trained and benchmarked, thereby assisting users to reason
1604 if the model is right for their purposes and if it can achieve its stated outcomes.
1605 Gebru et al. [133] also proposed datasheets, a standardised documentation format to
1606 describe the need for a particular data set, the information contained within it and
1607 what scenarios it should be used for, including legal or ethical concerns.

1608 However, while target audiences for these documents may be of a more technical
1609 AI level (i.e., the ML engineer), there is still no standardised communication format
1610 for application developers to reason about using particular IWSs, and the ramifica-
1611 tions this may have on the applications they write is not fully conveyed. Hence, our
1612 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 [397], Google Cloud Vision [422], Azure Computer Vision [436]. Specifically, we targeted the AWS DetectLabels endpoint [395], the Google Cloud Vision annotate:images endpoint [420] and Azure's analyze endpoint [437]. 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 [119] 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 [410, 411, 418, 431, 432, 484, 485] also exist, including other popular services used in Asia [409, 430, 450, 451] (some offering 3D image analysis [408]). 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 [217]. We have made our other two data sets available online ([413]). We collected results and their responses from each service’s API endpoint using a python script [417] 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

1692 **(6) Consistency of the top label’s confidence over time.** The top label of an
1693 image may remain static from one week to the next for the same service, but
1694 its confidence values may change with time. Therefore, there is inconsistency
1695 in the top label’s confidence due to model evolution.

1696 For the above aspects of consistency, we calculated the spread of variation for the
1697 top label’s confidences of each service for every 1 percent point; that is, the frequency
1698 of top label confidences within 100–99%, 99–98% etc. The consistency of top label’s
1699 and their confidences between each service was determined by intersecting the labels
1700 of each service per image and grouping the intersecting label’s confidences together.
1701 This allowed us to determine relevant probability distributions. For reproducibility,
1702 all quantitative analysis is available online [414].

1703 4.5 Findings

1704 4.5.1 Consistency of top labels

1705 4.5.1.1 Consistency across services

1706 Table 4.2 presents the consistency of the top labels between data sets, as measured
1707 by the cardinality of the intersection of all three services’ set of top labels divided
1708 by the number of images per data set. A combination of services present varied
1709 overlaps in their top labels; services A and C provide the best overlap for all three

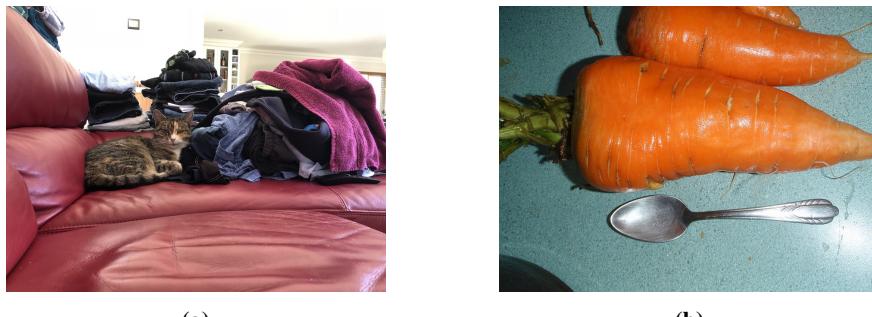


Figure 4.2: *Left:* The top labels for each service do not intersect, with each having a varied ontology: $T_i = \{ A = \{ \text{'black'} \}, B = \{ \text{'indoor'} \}, C = \{ \text{'slide'}, \text{'toy'} \} \}$. (Service C returns both ‘slide’ and ‘toy’ with equal confidence.) *Right:* The top labels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

1710 data sets, however the intersection of all three irrespective of data sets is low.

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

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

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



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.

1737 4.5.1.2 Semantic consistency where $|T_i| > 1$

1738 Service C returns two top labels for Figure 4.2a; ‘slide’ and ‘toy’. More than one
 1739 top label is typically returned in service C (80.00%, 56.97%, and 81.66% of all
 1740 images for all three data sets, respectively) though this also occurs in B in the large
 1741 (4.97% of all images) and COCOVal17 data sets (2.38%). Semantic inconsistencies
 1742 of what this label conceptually represents becomes a concern as these labels have
 1743 confidences of *equal highest* consistency. Thus, some services are inconsistent in
 1744 themselves and cannot give a guaranteed answer of what exists in an image; services
 1745 C and B have multiple top labels, but the respective services cannot ‘agree’ on
 1746 what the top label actually is. In Figure 4.3a, service C presents a reasonably high
 1747 confidence for the set of 7 top labels it returns, however there is too much diversity
 1748 ranging from a ‘hot chocolate’ to the hypernym ‘food’. Both are technically correct,
 1749 but it is up to the developer to decide the level of hypernymy to label the image as.
 1750 We also observe a similar effect in Figure 4.3b, where the image is labelled with
 1751 both the subject matter and the number of subjects per image.

1752 Thus, a taxonomy of ontologies is unknown; if a ‘border collie’ is detected in
 1753 an image, does this imply the hypernym ‘dog’ is detected, and then ‘mammal’, then
 1754 ‘animal’, then ‘object’? Only service B documents a taxonomy for capturing what
 1755 level of scope is desired, providing what it calls the ‘86-category’ concept as found
 1756 in its how-to guide:

1757 “*Identify and categorize an entire image, using a category taxonomy
 1758 with parent/child hereditary hierarchies. Categories can be used alone,
 1759 or with our new tagging models.*” [438]

1760 Thus, even if Rosa implemented conceptual similarity analysis for the image, the
 1761 top label set may not provide sufficient information to derive at a conclusive answer,
 1762 and if simply relying on only one label in this set, information such as the duplicity
 1763 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 $\gtrapprox 95\%$, when compared to $\gtrapprox 87\%$ and $\gtrapprox 93\%$ 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

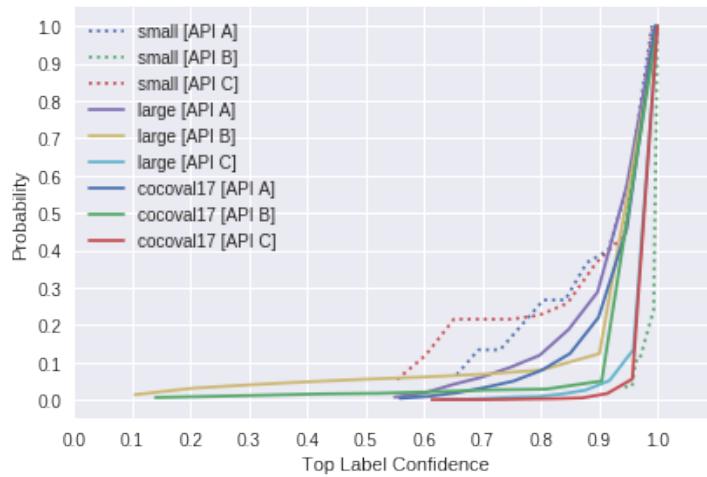


Figure 4.4: Cumulative distribution of the top labels' confidences. One in nine images return a top label(s) confident to $\gtrsim 97\%$, though there is a wider distribution for service A.

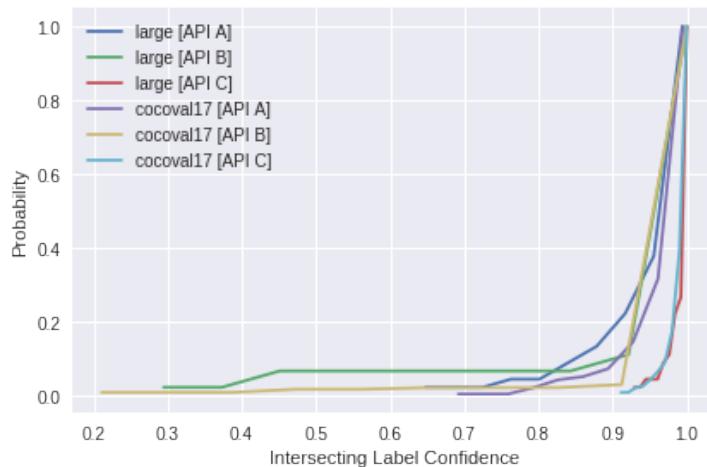


Figure 4.5: Cumulative distribution of intersecting top labels' confidences. The small data set is intentionally removed due to low intersections of labels (see Table 4.2).



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%.

¹⁷⁹⁴ deltas between snapshots (δ_c). However, this delta is so minuscule that we attribute
¹⁷⁹⁵ such changes to statistical noise.

¹⁷⁹⁶ 4.6 Recommendations

¹⁷⁹⁷ 4.6.1 Recommendations for IWS users

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

¹⁷⁹⁹ Rosa should ensure that in her testing strategies for the app she develops, there is an
¹⁸⁰⁰ ontology focus for the types of vocabulary that are returned. Additionally, we noted
¹⁸⁰¹ that there was a sudden change in case for services A and C; for all comparative
¹⁸⁰² purposes of labels, each label should be lower-cased.

¹⁸⁰³ 4.6.1.2 *Incorporate a specialised IWS testing methodology into the development 1804 lifecycle*

¹⁸⁰⁵ Rosa can utilise the different aspects of consistency as outlined in this paper as
¹⁸⁰⁶ part of her quality strategy. To ensure results are correct over time, we recommend
¹⁸⁰⁷ developers create a representative data set of the intended application’s data set
¹⁸⁰⁸ and evaluate these changes against their chosen service frequently. This will help
¹⁸⁰⁹ identify when changes, if any, have occurred if vendors do not provide a line of
¹⁸¹⁰ communication when this occurs.

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

¹⁸¹² Rosa will be locked into whichever vendor she chooses as there is inherent incon-
¹⁸¹³ sistency between these services in both the vocabulary and ontologies that they use.
¹⁸¹⁴ We have demonstrated that very few services overlap in their vocabularies, chiefly
¹⁸¹⁵ because they are still in early development and there is yet to be an established,
¹⁸¹⁶ standardised vocabulary that can be shared amongst the different vendors. Issues
¹⁸¹⁷ such as those shown in Section 4.5.1 can therefore be avoided.

1818 Throughout this work, we observed that the terminologies used by the vari-
1819 ous vendors are different. Documentation was studied, and we note that there is
1820 inconsistency between the ways techniques are described to users. We note the
1821 disparity between the terms ‘detection’, ‘recognition’, ‘localisation’ and ‘analysis’.
1822 This applies chiefly to object- and facial-related techniques. Detection applies to
1823 facial detection, which gives bounding box coordinates around all faces in an image.
1824 Similarly, localisation applies the same methodology to disparate objects in an image
1825 and labels them. In the context of facial ‘recognition’, this term implies that a face
1826 is *recognised* against a known set of faces. Lastly, ‘analysis’ applies in the context
1827 of facial analysis (gender, eye colour, expression etc.); there does not exist a similar
1828 analysis technique on objects.

1829 We notice similar patterns with object ‘tagging’, ‘detection’ and ‘labelling’.
1830 Service A uses ‘Entity Detection’ for object categorisation, service B uses ‘Image
1831 Tagging’, and service C uses the term ‘Detect Labels’ : conceptually, these provide
1832 the same functionality but the lack of consistency used between all three providers is
1833 concerning and leaves room for confusion with developers during any comparative
1834 analyses. Rosa may find that she wants to label her images into day/night scenes, but
1835 this in turn means the ‘labelling’ of varying objects. There is therefore no consistent
1836 standards to use the same terminology for the same concepts, as there are in other
1837 developer areas (such as Web Development).

1838 **4.6.1.4 Avoid use in safety-critical systems**

1839 We have demonstrated in this paper that both labels and confidences are stable but not
1840 constant; there is still an evolution risk posed to developers that may cause unknown
1841 consequences in applications dependent on these CVSs. Developers should avoid
1842 their use in safety critical systems due to the lack of visible changes.

1843 **4.6.2 Recommendations for IWS providers**

1844 **4.6.2.1 Improve the documentation**

1845 Rosa does not know that service A returns back ‘carrot’ for its top response, with
1846 service C returning ‘spoon’ (Figure 4.2b). She is unable to tell the service’s API
1847 where to focus on the image. Moreover, how can she toggle the level of specificity
1848 in her results? She is frustrated that service C can detect ‘chocolate’, ‘food’ and also
1849 ‘beverage’ all as the same top label in Figure 4.3a: what label is she to choose when
1850 the service is meant to do so for her, and how does she get around this? Thus, we
1851 recommend vendors to improve the documentation of services by making known
1852 the boundary set of the training data used for the algorithms. By making such
1853 information publicly available, developers would be able to review the service’s
1854 specificity for their intended use case (e.g., maybe Rosa is satisfied her app can
1855 catalogue ‘food’ together, and in fact does not want specific types of foods (‘hot
1856 chocolate’) catalogued). We also recommend that vendors publish usage guidelines
1857 should that include details of priors and how to evaluate the specific service results.

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

4.6.2.2 Improve versioning

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

4.6.2.3 Improve Metadata in Response

Much of the information in these services is reduced to a single confidence value within the response object, and the details about training data and the internal AI architecture remains unknown; little metadata is provided back to developers that encompass such detail. Early work into model cards and datasheets [133, 245] suggests more can be done to document attributes about ML systems, however at a minimum from our work, we recommend including a reference point via the form of an additional identifier. This identifier must also permit the developers to submit the identifier to another API endpoint should the developer wish to find further characteristics about the AI empowering the IWS, reinforcing the need for those presented in model cards and datasheets. For example, if Rosa sends this identifier she receives in the response object to the IWS descriptor API, she could find out additional information such as the version number or date when the model was trained, thereby resolving potential evolution risk, and/or the ontology of labels.

4.6.2.4 Apply constraints for predictions on all inputs

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

1898 A further example is sending a one pixel image to the service, analogous to
1899 sending an empty file. When we uploaded a single pixel white image to service A,
1900 we received responses such as ‘microwave oven’, ‘text’, ‘sky’, ‘white’ and ‘black’
1901 with confidences ranging from 51–95%. Prior checks should be performed on all
1902 input data, returning an ‘insufficient information’ error where any input data is below
1903 the information of its training data.

1904 **4.7 Threats to Validity**

1905 **4.7.1 Internal Validity**

1906 Not all CVSs were assessed. As suggested in Section 4.4, we note that there are
1907 other CVSs such as IBM Watson. Many services from Asia were also not considered
1908 due to language barriers (of the authors) in assessing these services. We limited our
1909 study to the most popular three providers (outside of Asia) to maintain focus in this
1910 body of work.

1911 A custom confidence threshold was not set. All responses returned from each of
1912 the services were included for analysis; where confidences were low, they were still
1913 included for analysis. This is because we used the default thresholds of each API to
1914 hint at what real-world applications may be like when testing and evaluating these
1915 services.

1916 The label string returned from each service was only considered. It is common
1917 for some labels to respond back that are conceptually similar (e.g., ‘car’ vs. ‘automobile’)
1918 or grammatically different (e.g., ‘clothes’ vs. ‘clothing’). While we could have
1919 employed more conceptual comparison or grammatical fixes in this study, we chose
1920 only to compare lowercased labels and as returned. We leave semantic comparison
1921 open to future work.

1922 Only introductory analysis has been applied in assessing the documentation of
1923 these services. Further detailed analysis of documentation quality against a rigorous
1924 documentation quality framework would be needed to fortify our analysis of the
1925 evolution of these services’ documentation.

1926 **4.7.2 External Validity**

1927 The documentation and services do change over time and evolve, with many allowing
1928 for contributions from the developer community via GitHub. We note that our
1929 evaluation of the documentation was conducted on a single date (see Section 4.4)
1930 and acknowledge that the documentation may have changed from the evaluation date
1931 to the time of this publication. We also acknowledge that the responses and labelling
1932 may have evolved too since the evaluation period described and the date of this
1933 publication. Thus, this may have an impact on the results we have produced in this
1934 paper compared to current, real-world results. To mitigate this, we have supplied the
1935 raw responses available online [415].

1936 Moreover, in this paper we have investigated *computer vision* services. Thus,
1937 the significance of our results to other domains such as natural language processing

1938 or audio transcription is, therefore, unknown. Future studies may wish to repeat our
1939 methodology on other domains to validate if similar patterns occur; we remain this
1940 open for future work.

1941 4.7.3 Construct Validity

1942 It is not clear if all the recommendations proposed in Section 4.6 are feasible
1943 or implementable in practice. Construct validity defines how well an experiment
1944 measures up to its claims; the experiments proposed in this paper support our three
1945 hypotheses but these have been conducted in a clinical condition. Real-world case
1946 studies and feedback from developers and providers in industry would remove the
1947 controlled nature of our work.

1948 4.8 Conclusions & Future Work

1949 This study explored three popular CVSs over an 11 month longitudinal experiment
1950 to determine if these services pose any evolution risk or inconsistency. We find that
1951 these services are generally stable but behave inconsistently; responses from these
1952 services do change with time and this is not visible to the developers who use them.
1953 Furthermore, the limitations of these systems are not properly conveyed by vendors.
1954 From our analysis, we present a set of recommendations for both IWS vendors and
1955 developers.

1956 Standardised software quality models (e.g., [172]) target maintainability and
1957 reliability as primary characteristics. Quality software is stable, testable, fault
1958 tolerant, easy to change and mature. These CVSs are, however, in a nascent stage,
1959 difficult to evaluate, and currently are not easily interchangeable. Effectively, the
1960 IWS response objects are shifting in material ways to developers, albeit slowly, and
1961 vendors do not communicate this evolution or modify API endpoints; the endpoint
1962 remains static but the content returned does not despite the same input.

1963 There are many potential directions stemming from this work. To start, we plan
1964 to focus on preparing a more comprehensive datasheet specifically targeted at what
1965 should be documented to application developers, and not data scientists. Reapplying
1966 this work in real-world contexts, that is, to get real developer opinions and study
1967 production grade systems, would also be beneficial to understand these phenomena
1968 in-context. This will help us clarify if such changes are a real concern for developers
1969 (i.e., if they really need to change between services, or the service evolution has real
1970 impact on their applications). We also wish to refine and systematise the method
1971 used in this study and develop change detectors that can be used to identify evolution
1972 in these services that can be applied to specific ML domains (i.e., not just computer
1973 vision), data sets, and API endpoints, thereby assisting application developers in their
1974 testing strategies. Moreover, future studies may wish to expand the methodology
1975 applied by refining how the responses are compared. As there does not yet exist a
1976 standardised list of terms available between services, labels could be *semantically*
1977 compared instead of using exact matches (e.g., by using stem words and synonyms
1978 to compare similar meanings of these labels), similar to previous studies [265].

1979 This paper has highlighted only some high-level issues that may be involved
1980 in using these evolving services. The laws of software evolution suggest that for
1981 software to be useful, it must evolve [241, 353]. There is, therefore, a trade-off, as
1982 we have shown, between consistency and evolution in this space. For a component
1983 to be stable, any changes to dependencies it relies on must be communicated. We
1984 are yet to see this maturity of communication from IWS providers. Thus, developers
1985 must be cautious between integrating intelligent components into their applications
1986 at the expense of stability; as the field of AI is moving quickly, we are more likely to
1987 see further instability and evolution in IWSs as a consequence.

CHAPTER 5

1988

1989

1990

1991

Interpreting Pain-Points in Computer Vision Services[†]

1992 **Abstract** Intelligent web services (IWSs) are becoming increasingly more pervasive; ap-
1993 plication developers want to leverage the latest advances in areas such as computer vision
1994 to provide new services and products to users, and large technology firms enable this via
1995 RESTful APIs. While such APIs promise an easy-to-integrate on-demand machine in-
1996 telligence, their current design, documentation and developer interface hides much of the
1997 underlying machine learning techniques that power them. Such APIs look and feel like
1998 conventional APIs but abstract away data-driven probabilistic behaviour—the implications
1999 of a developer treating these APIs in the same way as other, traditional cloud services, such
2000 as cloud storage, is of concern. The objective of this study is to determine the various
2001 pain-points developers face when implementing systems that rely on the most mature of
2002 these intelligent web services, specifically those that provide computer vision. We use Stack
2003 Overflow to mine indications of the frustrations that developers appear to face when using
2004 computer vision services, classifying their questions against two recent classification tax-
2005 onomies (documentation-related and general questions). We find that, unlike mature fields
2006 like mobile development, there is a contrast in the types of questions asked by developers.
2007 These indicate a shallow understanding of the underlying technology that empower such
2008 systems. We discuss several implications of these findings via the lens of learning tax-
2009 onomies to suggest how the software engineering community can improve these services
2010 and comment on the nature by which developers use them.

5.1 Introduction

2011 The availability of recent advances in artificial intelligence (AI) over simple RESTful
2012 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.

2014 web services (IWSs) are AI components that abstract complex machine learning
2015 (ML) and AI techniques behind simpler API calls. In particular, they hide (either
2016 explicitly or implicitly) any data-driven and non-deterministic properties inherent
2017 to the process of their construction. The promise is that software engineers can
2018 incorporate complex machine learnt capabilities, such as computer vision, by simply
2019 calling an API end-point.

2020 The expectation is that application developers can use these AI-powered services
2021 like they use other conventional software components and cloud services (e.g., object
2022 storage like AWS S3). Furthermore, the documentation of these AI components is
2023 still anchored to the traditional approach of briefly explaining the end-points with
2024 some information about the expected inputs and responses. The presupposition
2025 is that developers can reason and work with this high level information. These
2026 services are also marketed to suggest that application developers do not need to fully
2027 understand how these components were created (i.e., assumptions in training data
2028 and training algorithms), the ways in which the components can fail, and when such
2029 components should and should not be used.

2030 The nuances of ML and AI powering IWSs have to be appreciated, as there are
2031 real-world consequences to software quality for applications that depend on them if
2032 they are ignored [88]. This is especially true when ML and AI are abstracted and
2033 masked behind a conventional-looking API call, yet the mechanisms behind the API
2034 are data-dependent, probabilistic and potentially non-deterministic [265]. We are
2035 yet to discover what long-term impacts exist during development and production due
2036 to poor documentation that do not capture these traits, nor do we know the depth of
2037 understanding application developers have for these components. Given the way AI-
2038 powered services are currently presented, developers are also likely to reason about
2039 these new services much like a string library or a cloud data storage service. That
2040 is, they may not fully consider the implications of the underlying statistical nature
2041 of these new abstractions or the consequent impacts on productivity and quality.

2042 Typically, when developers are unable to correctly align to the mindset of the
2043 API designer, they attempt to resolve issues by (re-)reading the API documentation.
2044 If they are still unable to resolve these issues on their own after some internet
2045 searching, they consider online discussion platforms (e.g., Stack Overflow, GitHub
2046 Issues, Mailing Lists) where they seek technological advice from their peers [4].
2047 Capturing what developers discuss on these platforms offers an insight into the
2048 frustrations developers face when using different software components as shown
2049 by recent works [39, 190, 308, 337, 369]. However, to our knowledge, no studies
2050 have yet analysed what developers struggle with when using the new generation of
2051 *intelligent* services. Given the re-emergent interest in AI and the anticipated value
2052 from this technology [222], a better understanding of issues faced by developers
2053 will help us improve the quality of services. Our hypothesis is that application
2054 developers do not fully appreciate the probabilistic nature of these services, nor do
2055 they have sufficient appreciation of necessary background knowledge—however, we
2056 do not know the specific areas of concern. The motivation for our study is to inform
2057 API designers on which aspects to focus in their documentation, education, and
2058 potentially refine the design of the end-points.

2059 This study involves an investigation of 1,825 Stack Overflow (SO) posts regarding
2060 one of the most mature types of IWSs—computer vision services (CVSs)—dating
2061 from November 2012 to June 2019. We adapt existing methodologies of prior SO
2062 analyses [39, 348] to extract posts related to CVSs. We then apply two existing SO
2063 question classification schemes presented at ICPC and ICSE in 2018 and 2019 [4, 40].
2064 These previous studies focused on mobile apps and web applications. Although not
2065 a direct motivation, our work also serves as a validation of the applicability of these
2066 two issue classification taxonomies [4, 40] in the context of IWSs (hence potential
2067 for generalisation). Additionally our work is the first—to our knowledge—to *test*
2068 the applicability of these taxonomies in a new study.

2069 The taxonomies in previous works focus on the specific aspects from the domain
2070 (e.g. API usage, specificity within the documentation etc.) and as such do not
2071 deeply consider the learning gap of an application developer. To explore the API
2072 learning implications raised by our SO analysis, we applied an additional lens of
2073 two taxonomies from the field of pedagogy. This was motivated by the need to offer
2074 an insight into the work needed to help developers learn how to use these relatively
2075 new services.

2076 The key findings of our study are:

- 2077 • The primary areas that developers raise as issues reflect a relatively primitive
2078 understanding of the underlying concepts of data-driven ML approaches used.
2079 We note this via the issues raised due to conceptual misunderstanding and
2080 confusion in interpreting errors,
- 2081 • Developers predominantly encounter a different distribution of issue types than
2082 were reported in previous studies, indicating the complexity of the technical
2083 domain has a non-trivial influence on intelligent API usage; and
- 2084 • Most of these issues can be resolved with better documentation, based on our
2085 analysis.

2086 The paper also offers a data-set as an additional contribution to the research
2087 community and to permit replication [416]. The paper structure is as follows:
2088 Section 5.2 provides motivational examples to highlight the core focus of our study;
2089 Section 5.3 provides a background on prior studies that have mined SO to gather
2090 insight into the software engineering community; Section 5.4 describes our study
2091 design in detail; Section 5.5 presents the findings from the SO extraction; Section 5.6
2092 offers an interpretation of the results in addition to potential implications that arise
2093 from our work; Section 5.7 outlines the limitations of our study; concluding remarks
2094 are given in Section 5.8.

2095 **5.2 Motivation**

2096 “Intelligent” services are often available as a cloud end-point and provide devel-
2097 opers a friendly approach to access recent AI/ML advances without being experts
2098 in the underlying processes. Figure 5.1 highlights how these services abstract
2099 away much of the technical know-how needed to create and operationalise these
2100 IWSs [268]. In particular, they hide information about the training algorithm and

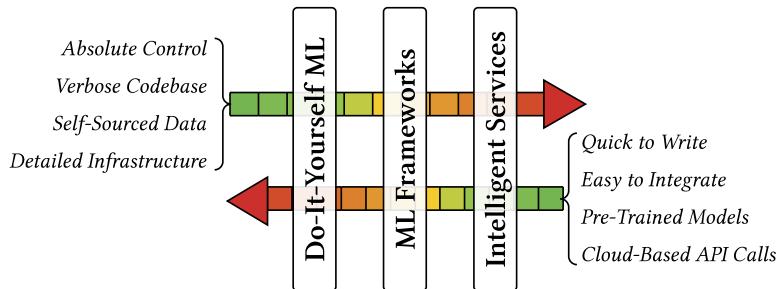


Figure 5.1: Some traits of Intelligent Services vs. ‘Do-It-Yourself’ ML. Green-to-red arrows indicate the presence of these traits. *Adapted from Ortiz [268].*

²¹⁰¹ data-sets used in training, the evaluation procedures, the optimisations undertaken,
²¹⁰² and—surprisingly—they often do not offer a properly versioned end-point [88, 265].
²¹⁰³ That is, the cloud vendors may change the behaviour of the services without sufficient
²¹⁰⁴ 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.” [461]; “I’m trying out the [IWS], and there’s a score field that returns that I’m not sure how to interpret [it].” [475]
 - **frustrated about non-deterministic results:** “Often the API has troubles in recognizing single digits... At other times Vision confuses digits with letters.” [474]; “Is there a way to help the program recognize numbers better, for example limit the results to a specific format, or to numbers only?” [471]
 - **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.” [455]
 - **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.” [458]

The objective of our study is to better understand the nature of the questions that developers raise when using IWSs, in order to inform the service designers and documenters. In particular, the knowledge we identify can be used to improve the documentation, educational material and (potentially) the information contained in the services' response objects—these are the main avenues developers have to learn and reason about when using these services. There is previous work that has investigated issues raised by developers [4, 40, 348]. 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:

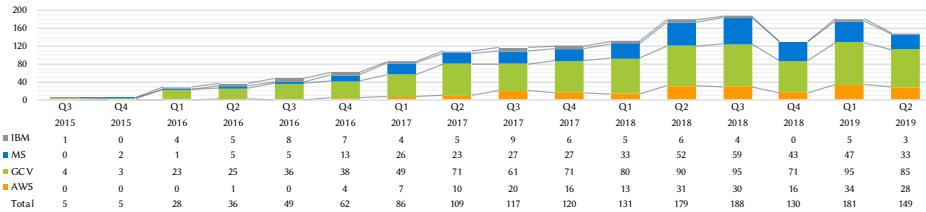


Figure 5.2: Trend of posts, where IBM = IBM Watson Visual Recognition, MS = Azure Computer Vision, AWS = AWS Rekognition and GCV = Google Cloud Vision. Three MS posts from Q4 2012, Q3 2013 and Q4 2013 have been removed for graph clarity.

- 2133 **RQ1. How do developers mis-comprehend IWSs as presented within SO**
 2134 **pain-points?** While the AI community is well aware in the nuances that
 2135 empower IWSs, such services are being released for application developers
 2136 who may not be aware of their limitations or how they work. This is
 2137 especially the case when machine intelligence is accessed via web-based
 2138 APIs where such details are not fully exposed.
- 2139 **RQ2. Are the distribution of issues similar to prior studies?** We compare
 2140 how the distributions of previous studies' of posts about conventional,
 2141 deterministic API services differ from those of IWSs. By assessing the
 2142 distribution of IWSs' issues against similar studies that focus on mobile
 2143 and web development, we identify whether a new taxonomy is needed
 2144 specific to AI-based services, and if gaps specific to AI knowledge exist
 2145 that need to be captured in these taxonomies.

2146 5.3 Background

2147 The primary goal of analysing issues is to better understand the root causes. Hence,
 2148 a good issue classification taxonomy should ideally capture the underlying causal
 2149 aspects (instead of pure functional groupings) [76]. Although this idea (of cause
 2150 related classification) is not new (Chillarege advocated for it in this TSE paper in
 2151 1992), this is not a universally followed approach when studying online discussions
 2152 and some recent works have largely classified issues into the “*what is*” and not
 2153 “*how to fix it*” [28, 39, 358]. They typically (manually) classify discussion into
 2154 either *functional areas* (e.g., Website Design/CSS, Mobile App Development, .NET
 2155 Framework, Java [28]) or *descriptive areas* (e.g., Coding Style/Practice, Problem/-
 2156 Solution, Design, QA [28, 358]). As a result, many of these studies do not give
 2157 us a prioritised means of targeted attack on how to *resolve* these issues with, for
 2158 example, improved documentation. Interestingly, recent taxonomies that studied SO
 2159 data (Aghajani et al. [4] and Beyer et al. [40]) were causal in nature and developed to
 2160 understand discussions related to mobile and web applications. However, issues that
 2161 arise when developers use IWSs have not been studied, nor do we know if existing
 2162 issue classification taxonomies are sufficient in this domain.

2163 Researchers studying APIs have also attempted to understand developer's opin-
 2164 ions towards APIs [358], categorise the questions they ask about these APIs [28,

2165 30, 40, 308], and understand API related documentation and usage issues [4, 5, 8,
2166 28, 163, 348]. These studies often employ automation to assist in the data analysis
2167 stages of their research. Latent Dirichlet Allocation [8, 28, 308, 358] is applied for
2168 topic modelling and other ML techniques such as Random Forests [40], Conditional
2169 Random Fields [5] or Support Vector Machines [40, 163] are also used.

2170 However, automatic techniques are tuned to classify into *descriptive* categories,
2171 that is, they help paint a landscape of *what is*, but generally do not address the
2172 causal factors to address the issues in great detail. For example, functional areas
2173 such as ‘Website Design’ [28], ‘User Interface’ [39] or ‘Design’ [359] result from
2174 such analyses. These automatic approaches are generally non-causal, making it hard
2175 to address reasons for *why* developers are asking such questions. However, not all
2176 studies in the space use automatic techniques; other studies employ manual thematic
2177 analysis [4, 30, 348] (e.g., card sorting) or a combination of both [39, 40, 308, 355].
2178 Our work uses a manual approach for classification, and we use taxonomies that
2179 are more causally aligned allowing our findings to be directly useful in terms of
2180 addressing the issues.

2181 Evidence-based software engineering [196] has helped shape the last 15 years
2182 worth of research, but the reliability of such evidence has been questioned [183, 185,
2183 328]. Replication studies, especially in empirical works, can give us the confidence
2184 that existing results are adaptable to new domains; in this context, we extend (to
2185 IWSs) and work with study methods developed in previous works.

2186 5.4 Method

2187 5.4.1 Data Extraction

2188 This study initially attempted to capture SO posts on a broad range of many IWSs by
2189 identifying issues related to four popular IWS cloud providers: Google Cloud [422],
2190 AWS [397], Azure [436] and IBM Cloud [432]. We based our selection criteria on
2191 the prominence of the providers in industry (Google, Amazon, Microsoft, IBM) and
2192 their ubiquity in cloud platform services. Additionally, in 2018, these services were
2193 considered the most adopted cloud vendors for enterprise applications [119].

2194 However, during the filtering stage (see Section 5.4.2), we decided to focus on
2195 a subset of these services, computer vision, as these are one of the more mature
2196 and stable ML/AI-based services with widespread and increasing adoption in the
2197 developer community (see Figure 5.2). We acknowledge other services beyond the
2198 four analysed provide similar capabilities [410, 411, 418, 431, 484, 485] and only
2199 English-speaking services have been selected, excluding popular services from Asia
2200 (e.g., [408, 409, 430, 450, 451])—see Section 5.7. For comprehensiveness, we
2201 explain below our initial attempts to extract *all* IWSs.

2202 5.4.1.1 Defining a list of IWSs

2203 As there exists no global ‘list’ of IWSs to search on, we needed to derive a *corpus*
2204 of *initial terms* to allow us to know *what* to search for on the Stack Exchange Data

2205 Explorer¹ (SEDE). We began by looking at different brand names of cloud services
2206 and their permutations (e.g., Google Cloud Services and GCS) as well as various
2207 ML-related products (e.g., Google Cloud ML). To do this, we performed extensive
2208 Google searches² in addition to manually reviewing six ‘overview’ pages of the
2209 relevant cloud platforms. We identified 91 initial IWSs to incorporate into our
2210 search terms³.

2211 5.4.1.2 *Manual search for relevant, related terms*

2212 We then ran a manual search² on each term to determine if these terms were relevant.
2213 We did this by querying each term within SO’s search feature, reviewing the titles
2214 and body post previews of the first three pages of results (we did not review the
2215 answers, only the questions). We also noted down the user-defined *Tags* of each post
2216 (up to five per question); by clicking into each tag, we could review similar tags (e.g.,
2217 ‘project-oxford’ for ‘azure-cognitive-services’) and check if the tag had synonyms
2218 (e.g., ‘aws-lex’ and ‘amazon-lex’). We then compiled a *corpus of tags* consisting of
2219 31 terms.

2220 5.4.1.3 *Developing a search query*

2221 We recognise that searching SEDE via *Tags* exclusively can be ineffective (see [28,
2222 348]). To mitigate this, we produced a *corpus of title and body terms*. Such terms
2223 are those that exist within the title and body of the posts to reflect the ways in
2224 which individual developers commonly use to refer to different IWSs. To derive
2225 at such a list, we performed a search^{2,3} of the 31 tags above in SEDE, filtering
2226 out posts that were not answers (i.e., questions only) as we wanted to see how
2227 developers *phrase* their questions. For each search, we extracted a random sample
2228 of 100 questions (400 total for each service) and reviewed each question. We noted
2229 many patterns in the permutations of how developers refer to these services, such
2230 as: common misspellings ('bind' vs. 'bing'); brand misunderstanding ('Microsoft
2231 computer vision' vs. 'Azure computer vision'); hyphenation ('Auto-ML' vs. 'Auto
2232 ML'); UK and US English ('Watson Analyser' vs. 'Watson Analyzer'); and, the
2233 use of apostrophes, plurals, and abbreviations ('Microsoft's Computer Vision API',
2234 'Microsoft Computer Vision Services', 'GCV' vs. 'Google Cloud Vision'). We
2235 arrived at a final list of 229 terms compromising all of the IWSs provided by
2236 Google, Amazon, Microsoft and IBM as of January 2019³.

2237 5.4.1.4 *Executing our search query*

2238 Our next step was to perform a case-insensitive search of all 229 terms within the
2239 body or title of posts. We used Google BigQuery’s public data-set of SO posts⁴ to
2240 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>

²²⁴¹ search was conducted on 10 May 2019, where we extracted 21,226 results. We then
²²⁴² performed several filtering steps to cleanse our extracted data, as explained below.

²²⁴³ 5.4.2 Data Filtering

²²⁴⁴ 5.4.2.1 Refining our inclusion/exclusion criteria

²²⁴⁵ We performed an initial manual filtering of the 50 most recent posts (sorted by
²²⁴⁶ descending *CreationDate* values) of the 21,226 posts above, assessing the suitability
²²⁴⁷ of the results and to help further refine our inclusion and exclusion criteria. We
²²⁴⁸ did note that some abbreviations used in the search terms (e.g., ‘GCV’, ‘WCS’⁵),
²²⁴⁹ resulting in irrelevant questions in our result set. We therefore removed abbreviations
²²⁵⁰ from our search query and consolidated all overlapping terms (e.g., ‘Google Vision
²²⁵¹ API’ was collapsed into ‘Google Vision’).

²²⁵² We also recognised that 21,226 results would be non-trivial to analyse without
²²⁵³ automated techniques. As we wanted to do manual qualitative analysis, we reduced
²²⁵⁴ our search space to 27 search terms of just the CVSs within the original corpus of
²²⁵⁵ 229 terms. These were Google Cloud Vision [422], AWS Rekognition [397], Azure
²²⁵⁶ Computer Vision [436], and IBM Watson Visual Recognition [432]. This resulted
²²⁵⁷ in 1,425 results that were extracted on 21 June 2019. The query used and raw results
²²⁵⁸ are available online in our supplementary materials [416].

²²⁵⁹ 5.4.2.2 Duplicates

²²⁶⁰ Within 1,425 results, no duplicate questions were noted, as determined by unique
²²⁶¹ post ID, title or timestamp.

²²⁶² 5.4.2.3 Automated and manual filtering

²²⁶³ To assess the suitability and nature of the 1,425 questions extracted, the first author
²²⁶⁴ began with a manual check on a randomised sample of 50 questions. As the questions
²²⁶⁵ were exported in a raw CSV format (with HTML tags included in the post’s body), we
²²⁶⁶ parsed the questions through an ERB templating engine script⁶ in which the ID, title,
²²⁶⁷ body, tags, created date, and view, answer and comment counts were rendered for
²²⁶⁸ each post in an easily-readable format. Additionally, SQL matches in the extraction
²²⁶⁹ process were also highlighted in yellow (i.e., in the body of the post) and listed at
²²⁷⁰ the top of each post. These visual cues helped to identify 3 false positive matches
²²⁷¹ where library imports or stack traces included terms within our corpus of 26 CVS
²²⁷² terms. For example, `aws-java-sdk-rekognition:jar` is falsely matched as a
²²⁷³ dependency within an unrelated question. As such exact matches would be hard to
²²⁷⁴ remove without the use of regular expressions, and due to the low likelihood (6%)
²²⁷⁵ 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>

2276 **5.4.2.4 Classification**

2277 Our 1,425 posts were then split into 4 additional random samples (in addition to the
2278 random sample of 50 above). 475 posts were classified by the first author and three
2279 other research assistants, software engineers with at least 2 years industry experience,
2280 assisted to classify the remaining 900. This left a total of 1,375 classifications
2281 made by four people plus an additional 450 classifications made from reliability
2282 analysis, in which the remaining 50 posts were classified nine times (as detailed in
2283 Section 5.4.3.1). Thus, a total of 1,825 classifications were made from the original
2284 1,425 posts extracted.

2285 Whilst we could have chosen to employ topic modelling, these are too descriptive
2286 in nature (as discussed in Section 5.3). Moreover, we wanted to see if prior
2287 taxonomies can be applied to IWSs (as opposed to creating a new one) and compare
2288 if their distributions are similar. Therefore, we applied the two existing taxonomies
2289 described in Section 5.3 to each post; (i) a documentation-specific taxonomy that ad-
2290 dresses issues directly resulting from documentation, and (ii) a generalised taxonomy
2291 that covers a broad range of SO issues in a well-defined software engineering area
2292 (specifically mobile app development). Aghajani et al.’s documentation-specific
2293 taxonomy (Taxonomy A) is multi-layered consisting of four dimensions and 16
2294 sub-categories [4]. Similarly, Beyer’s SO generalised post classification taxonomy
2295 (Taxonomy B) consists of seven dimensions [40]. We code each dimension with
2296 a number, X , and each sub-category with a letter y : (Xy). We describe both tax-
2297 onomies in detail within Table 5.1. Where a post was included in our results but
2298 not applicable to IWSs (see Section 5.4.2.3) or not applicable to a taxonomy dimen-
2299 sion/category, then the post was flagged for removal in further analysis. Table 5.1
2300 presents *our understanding* of the respective taxonomies; our intent is not to method-
2301 logically replicate Aghajani et al. or Beyer et al.’s studies in the IWS domain, rather
2302 to acknowledge related work in the area of SO classification and reduce the need to
2303 synthesise a new taxonomy. We baseline all coding against *our interpretation only*.
2304 Our classifications are therefore independent of the previous authors’ findings.

2305 **5.4.3 Data Analysis**

2306 **5.4.3.1 Reliability of Classification**

2307 To measure consistency of the categories assigned by each rater to each post, we
2308 utilised both intra- and inter-rater reliability [235]. As verbatim descriptions from
2309 dimensions and sub-categories were considered quite lengthy from their original
2310 sources, all raters met to agree on a shared interpretation of the descriptions, which
2311 were then paraphrased as discussed in the previous subsection and tabulated in
2312 Table 5.1. To perform statistical calculations of reliability, each category was as-
2313 signed a nominal value and a random sample of 50 posts were extracted. Two-phase
2314 reliability analysis followed.

2315 Firstly, intra-rater agreement by the first author was conducted twice on 28 June
2316 2019 and 9 August 2019. Secondly, inter-rater agreement was conducted with the
2317 remaining four co-authors in addition to three research assistants within our research

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

A Documentation-specific classification (Aghajani et al. [4])		
A-1	■ Information Content (What)	Issues related to what is written in the documentation
A-1a	→ <i>Correctness</i>	What exists in the documentation actually matches what is implemented in code
A-1b	→ <i>Completeness</i>	The documentation fully covers all aspects of the API's components
A-1c	→ <i>Up-to-dateness</i>	What is documented is accurate to the current version of the API
A-2	■ Information Content (How)	Issues related to how the document is written and organised
A-2a	→ <i>Maintainability</i>	The upkeep effort to ensure the documentation remains up to date
A-2b	→ <i>Readability</i>	The extent to which the documentation is interpretable
A-2c	→ <i>Usability</i>	How useable the organisation, look and feel of the documentation is
A-2d	→ <i>Usefulness</i>	The usefulness of the documentation, avoiding misinformation.
A-3	■ Process-Related	Issues related to the documentation process
A-3a	→ <i>Internationalisation</i>	Translating the documentation into other languages
A-3b	→ <i>Contribution-Related</i>	Contribution issues encountered when people contribute to the documentation
A-3c	→ <i>Configuration-Related</i>	Configuration issues of the documentation tool
A-3d	→ <i>Implementation-Related</i>	Unwanted development issues caused by (poor) documentation
A-3e	→ <i>Traceability</i>	Tracing documentation changes (when, when, who and why)
A-4	■ Tool-Related	Issues related to documentation tools (e.g., Javadoc)
A-4a	→ <i>Tooling Bugs</i>	Bugs that exist within the documentation tooling
A-3b	→ <i>Tooling Discrepancy</i>	Support as expectations not being fulfilled by these documentation tools
A-3c	→ <i>Tooling Help Required</i>	Help required due to improper usage of the tools
A-3d	→ <i>Tooling Migration</i>	Issues migrating the tool to a new version or another tool
B Generalised classification (Beyer et al. [40])		
B-1	■ API usage	Issue on how to implement something using a specific component provided by the API
B-2	■ Discrepancy	The questioner's <i>expected behaviour</i> of the API does not reflect the API's <i>actual behaviour</i>
B-3	■ Errors	Issue regarding some form of error when using the API, and provides an exception and/or stack trace to help understand why it is occurring
B-4	■ 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
B-5	■ 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
B-6	■ API change	Issue regarding changes in the API from a previous version
B-7	■ 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

2318 group in mid-August 2019. Thus, the 50 posts were classified an additional nine
2319 times, resulting in 450 classifications for reliability analysis. We include these
2320 classifications in our overall analysis.

2321 At first, we followed methods of reliability analysis similar to previous SO
2322 studies (e.g., [348]) using the percentage agreement metric that divides the number
2323 of agreed categories assigned per post by the total number of raters [235]. However,
2324 percentage agreement is generally rejected as an inadequate measure of reliability
2325 analysis [81, 149, 203] in statistical communities. As we used more than 2 coders
2326 and our reliability analysis was conducted under the same random sample of 50
2327 posts, we applied *Light's Kappa* [214] to our ratings, which indicates an overall
2328 index of agreement. This was done using the `irr` computational R package [128]
2329 as suggested in [149].

2330 **5.4.3.2 Distribution Analysis**

2331 In order to compare the distribution of categories from our study with previous studies
2332 we carried out a χ^2 test. We selected a χ^2 test as the following assumptions [329]
2333 are satisfied: (i) the data is categorical, (ii) all counts are greater than 5, and (iii)
2334 we can assume simple random sampling. The null hypothesis describes the case
2335 where each population has the same proportion of observations and the alternative
2336 hypothesis is where at least one of the null hypothesis statements is false. We chose
2337 a significance value, α , of 0.05 following a standard rule of thumb. As to the best
2338 of our knowledge this is the first statistical comparison using Taxonomy A and B on
2339 SO posts. To report the effect size we selected Cramer's Phi, ϕ_c which is well suited
2340 for use on nominal data [329].

2341 **5.5 Findings**

2342 We present our findings from classifying a total of 1,825 SO posts aimed at answering
2343 RQs 1 and 2. 450 posts were classified using Taxonomies A and B for reliability
2344 analysis as described in Section 5.4.3.1 and the remaining 1,375 posts were classified
2345 as per Section 5.4.2.4. A summary of our classification using Taxonomies A and B
2346 is shown in Figure 5.3.

2347 **5.5.1 Post classification and reliability analysis**

2348 When undertaking the classification, we found that 238 issues (13.04%) did not
2349 relate to IWSs directly. For example, library dependencies were still included in
2350 a number of results (see Section 5.4.2.3), and we found there to be many posts
2351 discussing Android's Mobile Vision API as Google (Cloud) Vision. These issues
2352 were flagged and ignored for further analysis (see Section 5.4.2.4).

2353 For our reliability analysis, we classified a total of 450 posts of which 70 posts
2354 were flagged as irrelevant. Landis and Koch [209] provide guidelines to interpret
2355 kappa reliability statistics, where $0.00 \leq \kappa \leq 0.20$ indicates *slight* agreement and
2356 $0.21 \leq \kappa \leq 0.40$ indicates *fair* agreement. Despite all raters meeting to agree

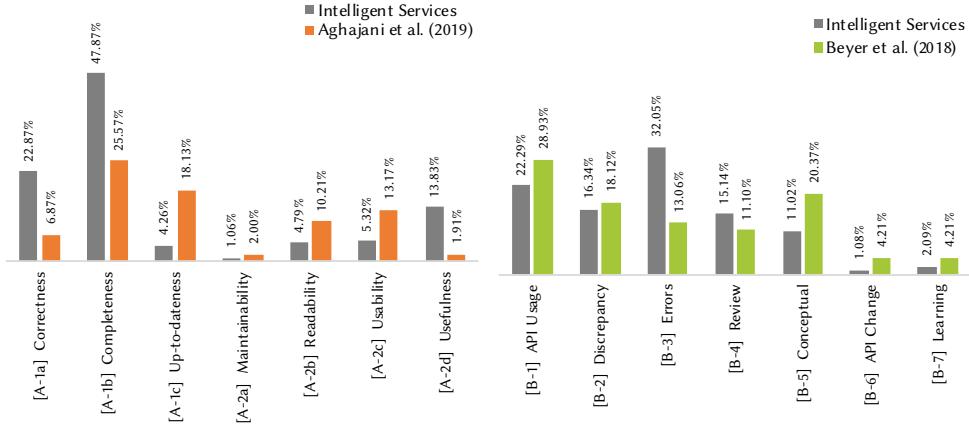


Figure 5.3: *Left:* Documentation-specific classification taxonomy results highlights a mostly similar distribution to that of Aghajani et al.’s findings [4]. *Right:* Generalised classification taxonomy results highlight differences from more mature fields (i.e., Android APIs in Beyer et al. [40]) to less mature fields (i.e., IWSs).

on a shared interpretation of the taxonomies (see Section 5.4.3.1) our inter-rater 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?*” [458]; “*There seems to be very little to no documentation for AWS iOS text recognition inside an image*” [456].

2381 22.87% of posts question the **accuracy (A-1a)** of certain parts of the cloud docu-
2382 mentation, especially in relation to incorrect quotas and limitations: “*Are the Cloud*
2383 *Vision API limits in documentation correct?*” [469], “*According to the Google Vision*
2384 *documentation, the maximum number of image files per request is 16. Elsewhere,*
2385 *however, I’m finding that the maximum number of requests per minute is as high as*
2386 *1800.*” [454].

2387 There are also many references (23.94%) addressing the confusing nature of
2388 some documentation, indicating that the **readability, usability and usefulness of**
2389 **the documentation (A-2b, A-2c and A-2d)** could be improved. For example, “*Am*
2390 *I encoding it correctly? The docs are quite vague.*” [452], “*The aws docs for this*
2391 *are really confusing.*” [481].

2392 5.5.2.2 Results from Beyer et al.’s taxonomy

2393 We found that a majority (32.05%) of posts are primarily **error-related questions**
2394 (**B-3**), including a dump of the stack trace or exception message from the service’s
2395 programming-language SDK (usually Java, Python or C#) that relates to a specific
2396 error. For example: “*I can’t fix an error that’s causing us to fall behind.*” [478]; “*I’m*
2397 *using the Java Google Vision API to run through a batch of images... I’m now getting*
2398 *a channel closed and ClosedChannelException error on the request.*” [472].

2399 **API usage questions (B-1)** were the second highest category at 22.29% of
2400 posts. Reading the questions revealed that many developers present an insufficient
2401 understanding of the behaviour, functional capability and limitation of these services
2402 and the need for further data processing. For example, while Azure provides an
2403 image captioning service, this is not universal to all CVSSs: “*In Amazon Rekognition*
2404 *for image processing how do I get the caption for an image?*” [463]. Similarly,
2405 OCR-related and label-related questions often indicate interest in cross-language
2406 translation, where a separate translation service would be required: “*Can Google*
2407 *Cloud Vision generate labels in Spanish via its API?*” [477]; “[*How can I] specify*
2408 *language for response in Google Cloud Vision API*” [464]; “[*When I request a text*
2409 *detection of an image, it gives only English Alphabet characters (characters without*
2410 *accents) which is not enough for me. How can I get the UTF-32 characters?*” [459].

2411 It was commonplace to see questions that demonstrate a lack of depth in under-
2412 standing and appreciating how these services work, instead posting simple debugging
2413 questions. For instance, in the 11.02% of **conceptual-related questions (B-5)** that
2414 we categorised, we noticed causal links to a misunderstanding (or lack of aware-
2415 ness) of the vocabulary used within computer vision. For example: “*The problem*
2416 *is that I need to know not only what is on the image but also the position of that*
2417 *object. Some of those APIs have such feature but only for face detection.*” [470];
2418 “[*I want to know if the new image has a face similar to the original image.... [the*
2419 *service] can identify faces, but can I use it to get similar faces to the identified face*
2420 *in other images?*” [462]. It is evident that some application developers are not aware
2421 of conceptual differences in computer vision such as object/face *detection* versus
2422 *localisation* versus *recognition*.

2423 In the 16.34% of **discrepancy-related questions (B-2)**, we see further unaware-

²⁴²⁴ ness from developers in how the underlying systems work. In OCR-related questions,
²⁴²⁵ developers do not understand the pre-processing steps required before an OCR is
²⁴²⁶ performed. In instances where text is separated into multiple columns, for example,
²⁴²⁷ text is read top-down rather than left-to-right and segmentation would be required
²⁴²⁸ to achieve the expected results. For example, “*it appears that the API is using some*
²⁴²⁹ *kind of logic that makes it scan top to bottom on the left side and moving to right*
²⁴³⁰ *side and doing a top to bottom scan.*” [476]; “*this method returns scanned text in*
²⁴³¹ *wrong sequence... please tell me how to get text in proper sequence.*” [482].

²⁴³² A number of **review-related questions (B-4)** (15.14%) seem to provide some
²⁴³³ further depth in understanding the context to which these systems work, where training
²⁴³⁴ data (or training stages) are needed to understand how inferences are made: “*How*
²⁴³⁵ *can we find an exhaustive list (or graph) of all logos which are effectively recognized*
²⁴³⁶ *using Google Vision logo detection feature?*” [480]; “*when object banana is detected*
²⁴³⁷ *with accuracy greater than certain value, then next action will be dispatched... how*
²⁴³⁸ *can I confidently define and validate the threshold value for each item?*” [466].

²⁴³⁹ **API change (B-6)** was shown in 1.08% of posts, with evolution of the services
²⁴⁴⁰ occurring (e.g., due to new training data) but not necessarily documented “*Recently*
²⁴⁴¹ *something about the Google Vision API changed... Suddenly, the API started to*
²⁴⁴² *respond differently to my requests. I sent the same picture to the API today, and I*
²⁴⁴³ *got a different response (from the past).*” [479].

²⁴⁴⁴ 5.5.3 Statistical Distribution Analysis

²⁴⁴⁵ We obtained the following results $\chi^2 = 131.86$, $\alpha = 0.05$, $p \text{ value} = 2.2 \times 10^{-16}$ and
²⁴⁴⁶ $\phi_c = 0.362$ from our distribution analysis with Taxonomy A to compare our study
²⁴⁴⁷ with that of Aghajani et al. [4]. Comparing our study to Beyer et al. [40] produced the
²⁴⁴⁸ following results $\chi^2 = 145.58$, $\alpha = 0.05$, $p \text{ value} = 2.2 \times 10^{-16}$ and $\phi_c = 0.252$.
²⁴⁴⁹ These results show that we are able to reject the null hypothesis that the distribution
²⁴⁵⁰ of posts using each taxonomy was the same as the comparison study. While there are
²⁴⁵¹ limited guidelines for interpreting ϕ_c when there is no prior information for effect
²⁴⁵² size [343], Sun et al. suggests the following: $0.07 \leq \phi_c \leq 0.20$ indicates a *small*
²⁴⁵³ effect, $0.21 \leq \phi_c \leq 0.35$ indicates a *medium* effect, and $0.35 > \phi_c$ indicates a *large*
²⁴⁵⁴ effect. Based on this criteria we obtained a *large* effect size for the documentation-
²⁴⁵⁵ specific classification (Taxonomy A) and a *medium* effect size for the generalised
²⁴⁵⁶ classification (Taxonomy B).

²⁴⁵⁷ 5.6 Discussion

²⁴⁵⁸ 5.6.1 Answers to Research Questions

²⁴⁵⁹ 5.6.1.1 How do developers mis-comprehend IWSs as presented within SO pain- ²⁴⁶⁰ points? (RQ1)

²⁴⁶¹ Upon meeting to discuss the discrepancies between our categorisation of IWS usage
²⁴⁶² SO posts, we found that our interpretations of the *posts themselves* were largely sub-
²⁴⁶³ jective. For example, many posts presented multi-faceted dimensions for Taxonomy

2464 B; Beyer et al. [40] argue that a post can have more than one question category and
2465 therefore multi-label classification is appropriate at times. We highlight this further
2466 in the threats to validity (Section 5.7).

2467 We have to define the context of IWSs to address RQ1. We use the concept
2468 of a “technical domain” [25] to define this context. A technical domain captures
2469 the domain-specific concerns that influence the non-functional requirements of a
2470 system [25]. In the context of IWSs, the technical domain includes exploration, data
2471 engineering, distributed infrastructure, training data, and model characteristics as
2472 first class citizens [25]. We would then expect to see posts on SO related to these
2473 core concerns.

2474 In Figure 5.3, for the documentation-specific classification, the majority of posts
2475 were classified as **Completeness (A1-b)** related (47.87%). An interpretation for this
2476 is that the documentation does not adequately cover the technical domain concerns.
2477 Comments by developers such as “*I'm searching for a list of all the possible image
2478 labels that the Google Cloud Vision API can return?*” [457] indicates the documentation
2479 does not adequately describe the training data for the API—developers do
2480 not know the required usage assumptions. Another quote from a developer, “*Can
2481 Google Cloud Vision generate labels in Spanish via its API? ... [Does the API]
2482 allow to select which language to return the labels in?*” [477] points to a lack of
2483 details relating to the characteristics of the models used by the API. It would seem
2484 that developers are unaware of aspects of the technical domain concerns.

2485 The next most frequent category is **Correctness (A-1a)** with 22.87% of posts. In
2486 the context of the technical domain there are many limits that developers need to be
2487 aware of: range and increments of a model score [88]; required data pre-processing
2488 steps for optimal performance; and features provided by the models (as explained in
2489 Section 5.5.2.2). Considering the relation between technical concerns and software
2490 quality, developers are right to question providers on correctness; “*Are the Cloud
2491 Vision API limits in documentation correct?*” [469].

2492 5.6.1.2 *Are the distribution of issues similar to prior studies? (RQ2)*

2493 Visual inspection of Figure 5.3 shows that the distributions for the documentation-
2494 specific classification and the generalised classification are different (compared to
2495 prior studies). As a sanity check we conducted a χ^2 test and calculated the effect
2496 size ϕ_c . We were able to reject the null hypothesis for both classification schemes,
2497 that the distribution of issues were the same as the previous studies (see Section 5.5).
2498 We now discuss the most prominent differences between our study and the previous
2499 studies.

2500 In the context of IWS SO posts, Taxonomy B suggests that Errors (B-3) are
2501 discussed most amongst developers. These results are in contrast to similar studies
2502 made in more *mature* API domains, such as Mobile Development [26, 27, 39, 40, 308]
2503 and Web Development [355]. Here, API Usage (B-1) is much more frequently
2504 discussed, followed by Conceptual (B-5), Discrepancy (B-2) and Errors (B-3). We
2505 argue in the following section that an improved developer understanding can be
2506 achieved by educating them about the IWS lifecycle and the ‘whole’ system that

2507 wraps such services.

2508 In the Android study API usage questions (B-1) were the highest category
2509 (28.93% compared to 22.29% in our study). As stated in the analysis of the Error
2510 questions this discrepancy could be due to the maturity of the domain. However,
2511 another explanation could be the scope of the two individual studies. Beyer et al. [40]
2512 used a broad search strategy consisting of posts tagged Android. This search term
2513 fetches issues related to the entire Android platform which is significantly larger
2514 than searching for computer vision APIs using 229 search terms. As a consequence
2515 of more posts and more APIs there would be use cases resulting in additional posts
2516 related to API Usage (B-1).

2517 Applying existing SO taxonomies allowed us to better understand the distribution
2518 of the issues across different domains. In particular, the issues raised around IWSs
2519 appear to be primarily due to poor documentation, or insufficient explanation around
2520 errors and limitations. Hence, many of the concerns could be addressed by adding
2521 more details to the end-point descriptions, and by providing additional information
2522 around how these services are designed to work.

2523 5.6.2 The Developer’s Learning Approach

2524 In this subsection, we offer an explanation as to why developers are complaining
2525 about certain things when trying to use IWSs on SO (RQ1), as characterised through
2526 the use of prior SO classification frameworks (RQ2). This is described through
2527 the theoretical lenses of two learning taxonomies: Bloom’s context complexity and
2528 intellectual ability taxonomy, and the Structure of the Observed Learning Outcome
2529 (SOLO) taxonomy (i.e., the nature by which developer’s learn). We argue that the
2530 issues with using IWSs relating to the lower-levels of these learning taxonomies
2531 are easily solvable by slight fixes and improvements to the documentation of these
2532 services. However, the higher dimensions of these taxonomies demand far more
2533 rigorous mitigation strategies than documentation alone (potentially more structured
2534 education). Thus, many of the questions posted are from developers who are *learning*
2535 to *understand* the domain of IWSs and AI, and (hence) both SOLO and Bloom’s
2536 taxonomies are applicable for this discussion—as described below within the context
2537 of our domain—as pedagogical aides.

2538 5.6.2.1 Bloom’s Taxonomy

2539 The cognitive domain under Bloom’s taxonomy [45] consists of six objectives.
2540 Within the context of IWSs, developers are likely to ask questions due to causal links
2541 that exist in the following layers of Bloom’s taxonomy: (i) *knowledge*, where the
2542 developer does not remember or know of the basic concepts of computer vision and
2543 AI (in essence, they may think that AI is as smart as a human); (ii) *comprehension*,
2544 where the developer does not understand how to interpret basic concepts, or they
2545 are mis-understanding how they are used in context; (iii) *application*, where the
2546 developer is struggling to apply existing concepts within the context of their own
2547 situation; (iv) *analysis*, where the developer is unable to analyse the results from IWSs
2548 (i.e., understand response objects); (v) *evaluation*, where the developer is unable to

²⁵⁴⁹ evaluate issues and make use of best-practices when using IWSs; and (vi) *synthesise*,
²⁵⁵⁰ where the developer is posing creative questions to ask if new concepts are possible
²⁵⁵¹ with CVSs.

²⁵⁵² 5.6.2.2 SOLO Taxonomy

²⁵⁵³ The SOLO taxonomy [41] consists of five levels of understanding. The causal
²⁵⁵⁴ links behind the SO questions we have found relate to the following layers of the
²⁵⁵⁵ SOLO taxonomy: (i) *pre-structural*, where the developer has a question indicating
²⁵⁵⁶ incompetence or has little understanding of computer vision; (ii) *uni-structural*,
²⁵⁵⁷ where the developer is struggling with one key aspect (i.e., a simple question about
²⁵⁵⁸ computer vision); (iii) *multi-structural*, where the developer is questioning multiple
²⁵⁵⁹ concepts (independently) to understand how to build their system (e.g., system
²⁵⁶⁰ integration with the IWS); (iv) *relational*, where the developer is comparing and
²⁵⁶¹ contrasting the best ways to achieve something with IWSs; and (v) *extended abstract*,
²⁵⁶² where the developer poses a question theorising, formulating or postulating a new
²⁵⁶³ 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.” [473]	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).” [453]	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?” [460]	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?” [467]	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?” [465]	Analysis	Relational

²⁵⁶⁴ 5.6.2.3 Aligning SO taxonomies to Bloom’s and SOLO taxonomies

²⁵⁶⁵ To understand our findings with the lenses of pedagogical aids, we aligned Tax-
²⁵⁶⁶ onomies A and B to Bloom’s and the SOLO taxonomies for a random sample of 50
²⁵⁶⁷ issues described in Section 5.4.3.1. To do this, we reviewed all 50 of these SO posted

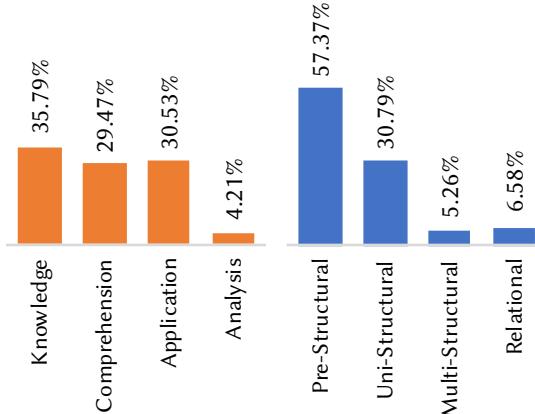


Figure 5.4: Alignment of Bloom (Orange) and SOLO (Blue) taxonomies against Taxonomy A and B dimensions against all 213 classifications made in the random sample of 50 posts.

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

As shown in Figure 5.4, the bulk of the posts fall in the lower constructs of Bloom’s and the SOLO taxonomy. This indicates that modification to certain documentation aspects can address many of these issues. For example, many issues can be ratified with better descriptions of response data and error messages: “*I was exploring google vision and in the specific function ‘detectCrops’, gives me the crop hints. what does this means exactly?*” [468]; “*I am making a very simple API call to the Google Vision API, but all the time it’s giving me error that ‘google.oauth2 module not found.’*” [483]

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

2591 5.6.3 Implications**2592 5.6.3.1 For Researchers**

2593 **Investigate the evolution of post classification** Analysing how the distribution of
2594 the reported issues changes over time would be an important study. This study could
2595 answer questions such as '*Does the evolution of IWSs follow the same pattern as*
2596 *previous software engineering trends such as mobile app or web development?*' As
2597 with any new emerging field, it is key to analyse how developers perceive such issues
2598 over time. For instance, early issues with web or mobile app development matured
2599 as their respective domain matured, and we would expect similar results to occur
2600 in the IWSs space. Future researchers could plan for a longitudinal study, such as
2601 a long-term survey with developers to gather their insights in this evolving domain,
2602 reviewing case studies of projects that use intelligent web services from now into
2603 the future, or re-mining SO at a later date and comparing the results to this study.
2604 This will help assess evolving trends and characteristics, and determine how and if
2605 the nature of the developer's experience with IWSs (and AI in general) changes with
2606 time.

2607 **Investigate the impact of technical challenges on API usage** As discussed above,
2608 IWSs have characteristics that may influence API usage patterns and should be
2609 investigated as a further avenue of research. Further mining of open source software
2610 repositories that make use of IWSs could be assessed, thereby investigating if API
2611 patterns evolve with the rise of AI-based applications.

2612 5.6.3.2 For Educators

2613 **Education on high-level aspects of IWSs** As demonstrated in our analysis of their
2614 SO posts, many developers appear to be unaware of the higher-level concepts that
2615 exist within the AI and ML realm. This includes the need to pre- and post-process
2616 data, the data dependency and instability that exists in these services, and the specific
2617 algorithms that empower the underlying intelligence and hence their limitations and
2618 characteristics. However, most developers don't seem to complain about these factors
2619 due to the lack of documentation (i.e., via Taxonomy A). Rather, they are unaware
2620 that such information should be documentation and instead ask generalised and open
2621 questions (i.e., via Taxonomy B). Thus, documentation improvements alone may not
2622 be enough to solve these issues. This results in uncertainty during the preparation
2623 and operation (usage) of such services. Such high-level conceptual information is
2624 currently largely missing in developer documentation for IWSs. Furthermore, many
2625 of the background ML and AI algorithm information needed to understand and use
2626 intelligent systems in context are built within data science (not software engineering)
2627 communities. A possible road-map to mitigate this issue would be the development
2628 of a software engineer's 'crash-course' in ML and AI. The aim of such a course
2629 would encourage software engineers to develop an appreciation of the nuances and
2630 the inherent risks and implications that comes with using IWSs. This could be
2631 taught at an undergraduate level to prepare the next generation of developers of a

‘programming 2.0’ era. However, the key aspects and implications that are presented with AI would need to be well-understood before such a course is developed, and determining the best strategy to curate the content to developers would be best left to the software engineering education domain. Further investigation in applying educational taxonomies in the area (such as our attempts to interpret our findings using Bloom’s and the SOLO taxonomies) would need to be thoroughly explored beforehand.

2639 5.6.3.3 For Software Engineers

2640 Better understanding of intelligent API contextual usage Our results show that
2641 developers are still learning to use these APIs. We applied two learning perspectives
2642 to interpret our results. In applying the two pedagogical taxonomies to our findings,
2643 we see that most issues seem to fall into the pre-structural and knowledge-based
2644 categories; little is asked of higher level concepts and a majority of issues do not
2645 offer complex analysis from developers. This suggests that developers are struggling
2646 as they are unaware of the vocabulary needed to actually use such APIs, further
2647 reinforcing the need for API providers to write overview documentation (as noted in
2648 prior work [87]) and not just simple endpoint documentation. This said, improved
2649 documentation isn’t always enough—as suggested by our discussion in Section 5.6.2,
2650 software engineers should explore further education to attain a greater appreciation
2651 of the nuances of ML when attempting to use these services.

2652 5.6.3.4 For Intelligent Service Providers

2653 Clarify use cases for IWSs Inspecting SO posts revealed that there is a level of
2654 confusion around the capabilities of different IWSs. This needs to be clarified in
2655 associated API documentation. The complication with this comes with targeting
2656 the documentation such that software developers (who are untrained in the nuances
2657 of AI and ML as per Section 5.6.3.2) can digest it and apply it in-context to
2658 application development.

2659 Technical domain matters More needs to be provided than a simple endpoint
2660 description as conventional APIs offer by describing the whole framework by which
2661 the endpoint sits, giving further context. This said, compared to traditional APIs,
2662 we find that developers complain less about the documentation and more about
2663 shallower issues. All expected pre-processing and post-processing needs to be
2664 clearly explained. A possible mitigation to this could be an interactive tutorial that
2665 helps developers fully understand the technical domain using a hands-on approach.
2666 For example, websites offer interactive Git tutorials⁷ to help developers understand
2667 and explore the technical domain matters under version control in their own pace.

2668 Clarify limitations API developers need to add clear limitations of the existing
2669 APIs. Limitations include list of objects that can be returned from an endpoint. We

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

2670 found that the cognitive anchors of how existing, conventional API documentation
2671 is written has become ‘ported’ to the computer vision realm, however a lot more
2672 overview documentation than what is given at present (i.e., better descriptions of
2673 errors, improved context of how these systems work in etc.) needs to be given. Such
2674 documentation could be provided using interactive tutorials.

2675 **5.7 Threats to Validity**

2676 **5.7.1 Internal Validity**

2677 As detailed in Section 5.4.3.1, Taxonomies A and B present slight and fair agreement,
2678 respectively, when inter-rater reliability was applied. The nature of our disagree-
2679 ments largely fell due to the subjectivity in applying either taxonomies to posts.
2680 Despite all coders agreeing to the shared interpretation of both taxonomies, both
2681 taxonomies are subjective in their application, which was not reported by either
2682 Aghajani et al. or Beyer et al.. In many cases, multi-label classification seemed ap-
2683 propriate, however both taxonomies use single-label mapping which we find results
2684 in too much subjectivity. This subjectivity, therefore, ultimately adversely affects
2685 inter-rater reliability (IRR) analysis. Thus, a future mitigation strategy for similar
2686 work should explore multi-label classification to avoid this issue; Beyer et al., for
2687 example, plan for multi-label classification as future work. However, these studies
2688 would need to consider the statistical challenges in calculating multi-rater, multi-
2689 label IRR for thorough reliability analysis in addressing subjectivity. The selection
2690 of SO posts used for our labelling, chiefly in the subjectivity of our classifications, is
2691 of concern. We mitigate this by an extensive review process assessing the reliability
2692 of our results as per Section 5.4.3.1. The classification of our posts into the SOLO
2693 and Bloom’s taxonomies was performed by the primary author only, and therefore
2694 no inter-rater reliability statistics were performed. However, we used these peda-
2695 gogy related taxonomies as a lens to gain an additional perspective to interpret our
2696 results. Future studies should attempt a more rigorous analysis of SO posts using
2697 Bloom’s and SOLO taxonomies. We only aligned posts to one category for each
2698 taxonomy and did not align these using multi-label classification. This brings more
2699 complexity to the analysis, and our attempts to repeat prior studies’ methodologies
2700 (see Section 5.3). Multi-label classification for IWSs SO posts is an avenue for future
2701 research.

2702 **5.7.2 External Validity**

2703 While every effort was made to select posts from SO relevant to CVSs, there are
2704 some cases where we may have missed some posts. This is especially due to the
2705 case where some developers mis-reference certain IWSs under different names (see
2706 Section 5.4.2.1).

2707 Our SOLO and Bloom’s taxonomy analysis has only been investigated through
2708 the lenses of IWSs, and not in terms of conventional APIs (e.g., Andriod APIs).
2709 Therefore, we are not fully certain how these results found would compare to other

2710 types of APIs. Two *existing* SO classification taxonomies were used rather than
2711 developing our own. We wanted to see if previous SO taxonomies could be applied
2712 to IWSs before developing a new, specific taxonomy, and these taxonomies were
2713 applied based on our interpretation (see Section 5.4.2.4) and may not necessarily
2714 reflect the interpretation of the original authors. Moreover, automated techniques
2715 such as topic modelling were not utilised as we found these produce descriptive
2716 classifications only (see Section 5.3). Hence, manual analysis was performed by
2717 humans to ensure categories could be aligned back to causal factors. Only English-
2718 speaking IWSs were selected; the applicability of our analysis to other, non-English
2719 speaking services may affect results. Use of computer vision in this study is an
2720 illustrative example to focus on one area of the IWSs spectrum. While our narrow
2721 scope helps us obtain more concrete findings, we suggest that wider issues exist in
2722 other IWS domains may affect the generalisability of this study, and suggest future
2723 work be explored in this space.

2724 5.7.3 Construct Validity

2725 Some questions extracted from SO produced false positives, as mentioned in Sec-
2726 tions 5.4.2.1, 5.4.2.3 and 5.5. However, all non-relevant posts were marked as noise
2727 for our study, and thus did not affect our findings. Moreover, SO is known to have
2728 issues where developers simply ask basic questions without looking at the actual
2729 documentation where the answer exists. Such questions, although down-voted, were
2730 still included in our data-set analysis, but as these were SO few, it does not have a
2731 substantial impact on categorised posts.

2732 5.8 Conclusions

2733 CVSs offer powerful capabilities that can be added into the developer’s toolkit via
2734 simple RESTful APIs. However, certain technical nuances of computer vision
2735 become abstracted away. We note that this abstraction comes at the expense of a full
2736 appreciation of the technical domain, context and proper usage of these systems. We
2737 applied two recent existing SO classification taxonomies (from 2018 and 2019) to see
2738 if existing taxonomies are able to fully categorise the types of complaints developers
2739 have. IWSs have a diverging distribution of the types of issues developers ask
2740 when compared to more mature domains (i.e., mobile app development and web
2741 development). Developers are more likely to complain about shallower, simple
2742 debugging issues without a distinct understanding of the AI algorithms that actually
2743 empower the APIs they use. Moreover, developers are more likely to complain about
2744 the completeness and correctness of existing IWS documentation, thereby suggesting
2745 that the documentation approach for these services should be reconsidered. Greater
2746 attention to education in the use of AI-powered APIs and their limitations is needed,
2747 and our discussion offered in Section 5.6.2 motivates future work in resolving these
2748 issues in the software engineering education space.

CHAPTER 6

2749

2750

2751

Ranking Computer Vision Service Issues using Emotion[†]

2752

2753 **Abstract** Software developers are increasingly using intelligent web services to implement
2754 ‘intelligent’ features. However, studies show that incorporating machine learning into an
2755 application increases technical debt, creates data dependencies, and introduces uncertainty
2756 due to their non-deterministic behaviour. We know very little about the emotional state of
2757 software developers who have to deal with such issues; a reduced developer experience when
2758 using such services results in productivity loss. In this paper, we conduct a landscape analysis
2759 of emotion found in 1,425 Stack Overflow questions about computer vision services. We
2760 used an existing emotion classifier, EmoTxt, and manually verified its classification results.
2761 We found that the emotion profile varies for different types of questions, and a discrepancy
2762 exists between automatic and manual emotion analysis due to subjectivity.

2763 6.1 Introduction

2764 Recent advances in artificial intelligence (AI) have provided software engineers
2765 with new opportunities to incorporate complex machine learning (ML) capabilities,
2766 such as computer vision, through cloud based intelligent web services (IWSs).
2767 These new set of services, typically offered as API calls are marketed as a way
2768 to reduce the complexity involved in integrating AI-components. However, recent
2769 work shows that software engineers struggle to use these IWSs [91]. Furthermore,
2770 the accompanying documentation fails to address common issues experienced by
2771 software engineers and often, engineers resort to online communication channels,
2772 such as Stack Overflow (SO), to seek advice from their peers [91].

[†]This chapter is originally based on A. Cummaudo, U. Graetsch, M. Curumsing, S. Barnett, R. Vasa, and J. Grundy, “Manual and Automatic Emotion Analysis of Computer Vision Service Pain-Points,” in *Proceedings of the Sixth International Workshop on Emotion Awareness in Software Engineering*. Virtual Event, USA: IEEE, 2021, In Review. Terminology has been updated to fit this thesis.

2773 While seeking advice on the issues, software engineers tend to express their
2774 emotions (such as frustration or confusion) within the questions. Emotions with
2775 negative sentiment have been shown to have adverse effects to developer productivity,
2776 as shown in [384], and thus—recognising the value of considering emotions—other
2777 literature has investigated how emotions are expressed by software developers within
2778 communication channels [269] including SO [68, 262]. The broad motivation
2779 of these works is to generally understand the emotional landscape and improve
2780 developer productivity [127, 248, 269]. However, previous works have not directly
2781 focused on the nature of emotions expressed in questions related to IWSs. We
2782 also do not know if certain types of questions express stronger emotions. Thus,
2783 understanding the emotional state of developers facing issues with these services
2784 can help shed light into prioritised choices of these services, avoid common issues
2785 which are the most frustrating (and thus highest productivity loss), and ultimately
2786 improve the developer experience (DevX) whilst using these services.

2787 The machine-learnt behaviour of these cloud IWSs is typically non-deterministic
2788 and, given the dimensions of data used, their internal inference process is hard to
2789 reason about [88]. Compounding the issue, documentation of these cloud systems
2790 does not explain the limits, nor how they were created (esp. information about the
2791 datasets used to train them). This lack of transparency makes it difficult for even
2792 senior developers to properly reason about these systems, so their prior experience
2793 and anchors do not offer sufficient support [91]. In addition, adding machine
2794 learned behaviour to a system incurs ongoing maintenance concerns [320]. There is
2795 a need to better understand emotions expressed by developers; as reduced negative
2796 emotions whilst writing software can improve productivity [384] and DevX, we can
2797 use such insight to help cloud vendors make improvement which would generate
2798 the most value, e.g., overall service/API design, documentation of the services or
2799 clarification in error messages.

2800 In our recent work [91], we explored the types of pain-points developers face
2801 when using IWSs through a general analysis of 1,425 SO questions using an existing
2802 SO question type classification taxonomy [40] (presented in Table 6.1). This study
2803 extends this body of work by considering the *emotional state* expressed within
2804 those same pain-points. We identify the emotion(s) in each SO question (if any),
2805 and investigate if the distribution of these emotions is similar across the various
2806 types of questions. To automate classification of these emotions, we used EmoTxt,
2807 an emotion classifier included in the EMTk toolkit for emotion recognition from
2808 text [67, 68, 262]. EmoTxt has been trained and built on SO posts using the emotion
2809 classification model proposed by Shaver et al. [326]. Additionally, we manually
2810 classified a sample of 300 posts using the same guidelines used to train EmoTxt,
2811 provided in [262] (based on [326]). The key contributions of this study are:

- 2812 • Identifying that the distribution of emotions is different across the taxonomy
2813 of issues.
- 2814 • A deeper analysis of the results obtained from the EmoTxt classifier suggests
2815 that the classification model needs further refinement. *Love* and *joy*, the
2816 least expected emotions when discussing API issues, are visible across all
2817 categories.

- 2818 • In order to promote future research and permit replication, we make our dataset
2819 publicly available.¹

2820 The paper is structured as follows: Section 6.2 provides an overview on prior
2821 work surrounding the classification of emotions from text; Section 6.3 describes our
2822 research methodology; Section 6.4 presents the results from the EmoTxt classifier;
2823 Section 6.5 provides a discussion of the results obtained; Section 6.6 outlines the
2824 threats to validity; Section 6.7 presents the concluding remarks.

2825 6.2 Motivation

2826 Developing software raises various emotions in developers at different times, includ-
2827 ing enjoyment, frustration, satisfaction, even fear and rage [67, 269, 384, 385]

2828 Studies on the role of emotions within the workplace, including the software
2829 engineering domain, have established a correlation between emotion and productiv-
2830 ity [384, 385]. Negative emotions impact productivity negatively, whilst positive
2831 emotions impact positively. The exception is *anger*, which was found to generate a
2832 motivating state to “try harder” in a subset of developers (i.e., 13% of respondents in
2833 Wrobel’s study [384] responded that anger had a *positive* impact to make developers
2834 more motivated. However, overall, *anger* was still found to have an negative impact
2835 on productivity). In recent years, researchers have focused on identifying the emotions
2836 expressed by software engineers within communication channels such as JIRA
2837 to communicate with their peers [127, 248, 262, 269]. Most of these studies make
2838 use of one of the well established emotion classification framework during their
2839 emotion mining process. Murgia et al. [248] and Ortú et al. [269] investigated the
2840 emotions expressed by developers within an issue tracking system, such as JIRA, by
2841 labelling issue comments and sentences written by developers using Parrott’s emotion
2842 framework. Gachechiladze et al. [127] applied the Shaver’s emotion framework
2843 to detect anger expressed in comments written by developers in JIRA.

2844 The Collab team [67, 262] extended the work done by Ortú et al. [269] and
2845 developed an emotion mining toolkit, EmoTxt [67] based on a gold standard dataset
2846 collected from 4,800 SO posts (of type questions, question comments, answers and
2847 answer comments). 12 graduate computer science students were recruited as raters
2848 to manually annotate these 4,800 SO posts using the Shaver’s emotion model which
2849 consists of a tree-structured, three level, hierarchical classification of emotions. The
2850 top level consists of six basic emotions namely, love, joy, anger, sadness, fear and
2851 surprise [326]. The work conducted by the Collab team is most relevant to our
2852 study since their focus is on identifying emotion from SO posts and their classifier
2853 is trained on a large dataset of SO posts. Unlike their study, we focus on a single
2854 domain (computer vision services or CVSs) to analysing emotion, as opposed to
2855 a wide spectrum of domains. Further, we validate our work with a smaller group
2856 of people—diverse in age and cultural backgrounds—to gather a wider sense of
2857 emotion classification (i.e., due to the subjective nature of emotions). Lastly, in this

¹See <https://bit.ly/2RIGQ2N>.

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

²⁸⁵⁸ work, our intent is to analyse the questions only (not all types of posts) to understand
²⁸⁵⁹ the frustration faced at the time the developers face an issue with the service.

²⁸⁶⁰ 6.3 Methodology

²⁸⁶¹ 6.3.1 Dataset

²⁸⁶² This paper extends our existing work by utilising our previously curated dataset
²⁸⁶³ of 1,425 SO questions on four popular computer vision service (CVS) providers:
²⁸⁶⁴ Google Cloud Vision, Amazon Rekognition, Azure Computer Vision, and IBM
²⁸⁶⁵ Watson. Each question is assigned a question type per the taxonomy prescribed in
²⁸⁶⁶ Beyer et al. [40] (for reference, we provide our interpretation of this taxonomy within
²⁸⁶⁷ Table 6.1). For further details on how this dataset was produced, we refer to the
²⁸⁶⁸ original paper [91].

²⁸⁶⁹ After performing additional cleansing of this dataset (to remove noise), we
²⁸⁷⁰ performed *both* automatic and manual emotion classification based on Shaver et al.’s
²⁸⁷¹ emotion taxonomy [326]. Automatic emotion detection was performed using
²⁸⁷² the EmoTxt classifier, and manual classification was performed by three co-authors
²⁸⁷³ on a sample of 300 posts. We calculated the inter-rater reliability between EmoTxt
²⁸⁷⁴ and our manually classified questions in two ways: (i) to see the overall agreement
²⁸⁷⁵ between the three raters in applying the Shaver et al. emotions taxonomy, and
²⁸⁷⁶ (ii) to see the overall agreement with EmoTxt’s classifications. Additional dataset
²⁸⁷⁷ cleansing and results from manual and automatic emotion classification are available
²⁸⁷⁸ online at <https://bit.ly/2RIGQ2N>.

2879 6.3.2 Additional Dataset Cleansing

2880 As described in [91], the 1,425 questions extracted were split into 5 random samples.
2881 The first author classified the first sample of 475 questions, with three other research
2882 assistants² classifying the remaining 900 questions over samples of 300 posts. The
2883 remaining 50 posts were used for reliability analysis, whereby these 50 posts were
2884 classified nine times by various researchers in our group, resulting in a total of 450
2885 classifications for the 50 posts.

2886 Each question was classified a question issue type (as described by Table 6.1) or,
2887 where the question was a false-positive resulting from our original search query, we
2888 flagged the post as ‘noise’ and removed them from further classification. 186 posts
2889 were flagged as noise, with a total of 1,239 were successfully assigned a question
2890 type.

2891 To remove duplicity resulting from the reliability analysis, we applied a ‘majority
2892 rules’ technique to each of these 50 posts, in which the issue type most consistent
2893 amongst the nine raters per question would win. As an example, three raters classified
2894 a post as *API Usage*, one rater classified the same post as a *Review* question and five
2895 raters classified the post as *Conceptual*. Therefore, the question was assigned as a
2896 *Conceptual* question. However, in four cases, there was a tie in the majority. To
2897 resolve this, we used the issue type that was most assigned within the 50 posts. For
2898 example, in another question, three raters each assigned the same post as *Discrepancy*
2899 and *Errors*, while the remaining three raters flagged the post as noise. In this
2900 case, the tie was resolved down to *Errors* as this classification received 72 more
2901 votes than *Discrepancy* and 88 more votes than noisy posts across all classifications
2902 made in the sample of 50 posts.

2903 6.3.3 Automatic Emotion Classification

2904 After all questions had been classified an issue type, we then piped in the body
2905 of each question into a script developed to remove all HTML tags, code snippets,
2906 blockquotes and hyperlinks, as suggested by Novielli et al. [262]. We replicated and
2907 extended the study conducted by Novielli et al. [262] on our dataset consisting of
2908 questions only. We started with a file containing the 1,239 non-noise SO questions,
2909 each with its associated question type given in Table 6.1. We pre-processed this file
2910 by extracting the question ID and body text to meet the format requirements of the
2911 EmoTxt classifier [67]. This classifier was used as it was trained on SO posts as
2912 discussed in Section 6.2. We ran the classifier for each emotion as this was required
2913 by EmoTxt model. This resulted in six output prediction files (one file for each
2914 emotion: *Love*, *Joy*, *Surprise*, *Sadness*, *Fear*, *Anger*), which referenced a question
2915 ID and a binary value indicating emotion presence. We then merged these emotion
2916 prediction files into an aggregate file with question text and Beyer et al.’s question
2917 type classifications that was performed in [91].

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

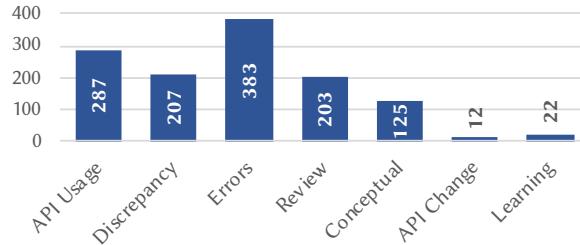


Figure 6.1: Distribution of the types of questions raised.

2918 6.3.4 Manual Emotion Classification

2919 In order to evaluate and also better understand the process used by EmoTxt to classify
 2920 emotions, we randomly sampled 300 SO posts of various emotion annotations resulting
 2921 from EmoTxt. Each of these 300 posts were assigned to three raters (co-authors
 2922 of this paper) who individually reviewed the question text against each of the six
 2923 basic emotions [326] and flag an emotion if deemed present, otherwise flagging *No*
 2924 *Emotion* instead. Each rater reviewed each question against the guidelines provided
 2925 in [262]. We then conducted reliability analysis of all three rater's results to measure
 2926 the similarity in which independent raters classified each emotions against each SO
 2927 post. This was done by calculating Cohen's Kappa (C_k) [81] to measure the average
 2928 inter-rater agreement *between* pairs of raters, and then Light's Kappa (L_k) [214]
 2929 to measure the *overall* agreement amongst the three raters. Results are reported in
 2930 Table 6.3.

2931 6.3.5 Comparing Manual and Automatic Classification Methods

2932 The next step involved comparing the ratings of the 300 SO posts that were manually
 2933 annotated by the three raters against the results obtained for the same set of 300 SO
 2934 posts from the EmoTxt classifier. We separated the classifications per emotion and
 2935 calculated C_k for each rater against EmoTxt, and then L_k to measure the overall
 2936 agreement. The three raters then met together to compare and discuss the ratings
 2937 from the EmoTxt classifier against the manual ratings. Results are reported in
 2938 Table 6.3.

2939 6.4 Findings

2940 Figure 6.1 displays the overall distribution of question types from the 1,239 posts
 2941 after applying noise-filtering and majority ruling to our original 1,425 questions
 2942 extracted. It is evident that developers ask issues predominantly related to API errors
 2943 when using CVSSs and, additionally, how they can use the API to implement specific
 2944 functionality. There are few questions related to version issues or self-learning. For
 2945 further discussion into these results, we refer to [91].

2946 Table 6.2 displays the frequency of questions that were classified by EmoTxt
 2947 when compared to our assignment of question types. Figure 6.2 presents the emotion

Table 6.2: Frequency of emotions per question type.

Question Type	Fear	Joy	Love	Sadness	Surprise	Anger	No Emotion	Total
API Usage	47	22	34	17	59	13	136	328
Discrepancy	35	12	17	7	46	20	105	242
Errors	73	34	23	21	47	23	207	428
Review	35	16	15	16	42	14	95	233
Conceptual	27	9	10	8	21	5	61	141
API Change	4	2	2	1	1	1	5	16
Learning	3	4	2	0	4	0	11	24
Total	224	99	103	70	220	76	620	1412

2948 data proportionally across each type of question. In total, 792 emotions were detected
2949 within the 1,239 non-noisy posts, and 620 questions where EmoTxt predicted *No*
2950 *Emotion* for all the emotion classification runs. Of the 792 questions with emotion
2951 detected, 114 questions had two emotions predicted, 28 questions had three emotions
2952 detected, and one question³ had four emotions detected (*Surprise*, *Sadness*, *Joy* and
2953 *Fear*).

2954 *No Emotion* was the most prevalent across all question types, which is consistent
2955 with the findings of the Collab group during the training of the EmoTxt classifier.
2956 Questions classified as *API Change* had the broadest distribution of emotions, with
2957 EmoTxt reporting 31.25% of these types of questions as *No Emotion*, compared to
2958 overall average of 42.10%. However, this is likely due to the low sample size of
2959 *API Change* questions (with only 12 questions assigned this issue type). The next
2960 highest set of emotive questions are found in the second and fourth largest samples
2961 (*Review* at 203 posts, and *API Usage* at 287 posts); therefore, higher proportions of
2962 emotion is not necessarily correlated to sample size.

2963 Unsurprisingly, *Discrepancy*-based questions—indicative of the frustrations de-
2964 velopers face when the API does something unexpected—had the highest proportion
2965 of *Anger* detected, at 8.26%, compared to *Anger*'s mean of 4.77%. To our surprise,
2966 *Love* (which we expected least by software developers when encountering issues)
2967 was present across all of the different question types. On average, this was reported at
2968 8.15%. The two highest emotions, by average, were *Fear* ($\mu = 16.77\%$) and *Surprise*
2969 ($\mu = 14.82\%$). In contrast, to our surprise, the two least-detected emotions reported
2970 by EmoTxt were *Sadness* ($\mu = 4.53\%$) and *Anger* ($\mu = 4.77\%$). *Joy* and *Love* were
2971 roughly the same, and fell in between the two proportion ends, with means of 8.85%
2972 and 8.15%, respectively.

2973 As shown in Table 6.3, results from our reliability analysis between human raters
2974 indicated subjectivity in emotion interpretation. Guidelines of indicative strengths
2975 of agreement are provided by Landis and Koch [209], where $\kappa \leq 0.00$ is *poor*
2976 agreement, $0.00 < \kappa \leq 0.20$ is *slight* agreement and $0.20 < \kappa \leq 0.40$ is *fair*
2977 agreement. Our assessments across the 300 questions indicate slight agreement for
2978 *Love*, *Surprise*, *Sadness*, *Anger* and *No Emotion*, and fair agreement for *Joy* and
2979 *Fear*. When combining human raters and EmoTxt, the inter-rater agreement was

³See <http://stackoverflow.com/q/55464541>.

Table 6.3: Inter-rater agreement between humans ($R_{1..3}$) and EmoTxt (E) and indicative guidelines of strength.

Emotion	$C_k(R_1, R_2)$	$C_k(R_1, R_3)$	$C_k(R_2, R_3)$	$L_k(R_{1..3})$	$C_k(R_1, E)$	$C_k(R_2, E)$	$C_k(R_3, E)$	$L_k(R_{1..3}, E)$
Love	0.30 Fair	0.17 Slight	0.04 Slight	0.17 Slight	0.37 Fair	0.27 Fair	0.05 Slight	0.20 Slight
Joy	0.21 Fair	0.16 Slight	0.57 Fair	0.31 Fair	0.1 Slight	0.07 Slight	-0.01 Poor	0.18 Slight
Surprise	0.21 Fair	0.13 Slight	0.15 Slight	0.16 Slight	0.17 Slight	0.04 Slight	0.06 Slight	0.13 Slight
Sadness	0.11 Slight	0.05 Slight	0.01 Slight	0.05 Slight	0.09 Slight	0.04 Slight	0.02 Slight	0.05 Slight
Fear	0.19 Slight	0.22 Fair	0.36 Fair	0.26 Fair	-0.02 Poor	-0.06 Poor	0.01 Slight	0.12 Slight
Anger	0.19 Slight	0.19 Slight	0.07 Slight	0.15 Slight	0.13 Slight	0.16 Slight	0.03 Slight	0.13 Slight
No Emotion	0.30 Fair	0.16 Slight	0.09 Slight	0.18 Slight	0.25 Fair	0.06 Slight	0.04 Slight	0.15 Slight

2980 slight across all emotions.

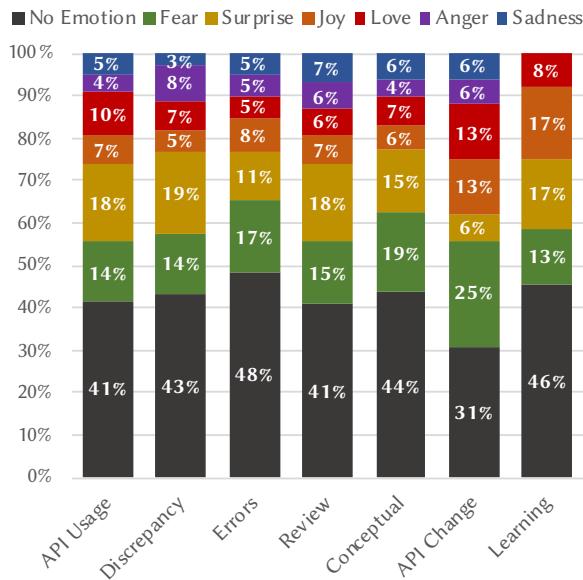


Figure 6.2: Proportion of emotions per question type.

2981 6.5 Discussion

2982 Our findings from the comparison between the manually annotated SO posts and the
 2983 automatic classification revealed substantial discrepancies. Table 6.4 provides some
 2984 sample questions⁴ from our dataset, with the Beyer et al. question classification type
 2985 noted with $[Q]$, the emotion(s) identified by EmoTxt within the text noted with $[E]$,
 2986 and the emotion(s) classified by the three raters indicated with $[R_{1..3}]$. The subset
 2987 of questions analysed by our three raters do not indicate the automatic (EmoTxt)
 2988 emotion, and upon manual inspection of the text after poor results from our reliability
 2989 analysis, an introspection of the dataset sheds some light to the discrepancy.

2990 For example, the first question in Table 6.4 shows no indication of *Joy*, but
 2991 EmoTxt classifies it to this emotion. Phrases like “*I’m pretty sure...*” could be the

⁴Questions located at [https://stackoverflow.com/q/\[ID\]](https://stackoverflow.com/q/[ID]).

Table 6.4: Sample of various question types ($[Q]$) against emotion(s) identified by EmoTxt ($[E]$) and the three raters ($[R_{1..3}]$).

Question ID and Quote	Classifications
51444352: “I’m pretty sure I set up my IAM role appropriately (I literally attached the ComprehendFullAccess policy to the role) and the Cognito Pool was also setup appropriately (I know this because I’m also using Rekognition and it works with the IAM Role and Cognito ID Pool I created) and yet every time I try to send a request to AWS Comprehend I get the error... Any idea of what I can do in this situation?”	$[Q]$: Errors $[E]$: Joy $[R_1]$: Surprise $[R_2]$: Surprise $[R_3]$: Anger
53117918: “Ok so I have been stuck here for about more than a week now and I know its some dumb mistake. Just can’t figure it out. I am working on a project that is available of two platforms, Android & iOS. Its sort of a facial recognition app... Is there anything I need to change? Is there any additional setup I need to do to make it work?Please let me know. Thanks.”	$[Q]$: Discrepancy $[E]$: Love, Surprise, Anger $[R_1]$: Sadness, Anger $[R_2]$: Sadness, Anger $[R_3]$: Anger
52829583: “I was trying to make the google vision OCR regex searchable... it fails when there is the text of other languages.It’s happening because I have only English characters in google vision word component as follows.As I can’t include characters from all the languages, I am thinking to include the inverse of above... So where can I find ALL THE SPECIAL CHARACTERS WHICH ARE IDENTIFIED AS A SEPARATE WORD BY GOOGLE VISION? Trial and error, keep adding the special characters I find is one option. But that would be my last option.”	$[Q]$: Review $[E]$: Anger $[R_1]$: Joy, Anger $[R_2]$: Anger $[R_3]$: Surprise
50190527: “I am trying to perform OCR on pdf documents using google cloud vision API, i uploaded a pdf document into a cloud bucket and downloaded the oauth key file and added it in the script as below. But when i run the file, i get the permission denied: 403 error, can anyone please give me instructions on how to fix it, i did extensive google search and did not yield any results, i am surely missing something here... I have checked the older stack overflow questions and the links provided in answers are not active anymore.Thanks in advance for your help.”	$[Q]$: API Usage $[E]$: No Emotion $[R_1]$: Sadness $[R_2]$: No Emotion $[R_3]$: Anger
52126752: “I am trying to call google cloud vision api from xamarin C# android application code.I have set environment variable but still I was not able to call api.So I decided to call it by passing credential json file but now I am getting error deserializing JSON credential datahere is my code”	$[Q]$: Errors $[E]$: Surprise $[R_1]$: No Emotion $[R_2]$: No Emotion $[R_3]$: Anger
48145425: “I am Deploying Google cloud vision Ocr in My angular2 webapp. but i am getting many of the errors when i add this code in my webapp code. please help me to sort out this.”	$[Q]$: Errors $[E]$: Fear $[R_1]$: Fear $[R_2]$: No Emotion $[R_3]$: Sadness

2992 reason why poor classification occurred, where words like “pretty” are associated
2993 with *Joy*, albeit in completely different context. It seems likely that the developer is
2994 experiencing a confusing situation when the API throws unexpected errors; thus [R_1]
2995 and [R_2] noting *Surprise*. Similarly, in the second question presented in Table 6.4,
2996 EmoTxt classifies *Love*, *Surprise*, and *Anger*. It is difficult to find an element of love
2997 or appreciation elsewhere in this context beyond the closing remarks: “**Please let me**
2998 **know. Thanks.**”. Moreover, the disparity between EmoTxt and the agreed emotions
2999 between the first two reviewers shows that EmoTxt cannot detect the frustration
3000 (*Anger*) in the developer’s tone, which is evident in their opening sentence, “*I have*
3001 *been stuck here for about more than a week and I know it is some dumb mistake.*”.

3002 These results indicate that introspection into the behaviour and limitations of
3003 the EmoTxt model is necessary. Our results indicate further work is needed to
3004 refine the ML classifiers that mine emotions in the SO context. The question that
3005 arises is whether the classification model is truly reflective of real-world emotions
3006 expressed by software developers. As highlighted by Curumsing [93], the divergence
3007 of opinions with regards to the emotion classification model proposed by theorists
3008 raises doubts to the foundations of basic emotions. Most of the studies conducted in
3009 the area of emotion mining from text is based on an existing general purpose emotion
3010 framework from psychology [63, 262, 269]—none of which are finely tuned for the
3011 software engineering domain.

3012 6.6 Threats to Validity

3013 6.6.1 Internal validity

3014 The *API Change* and *Learning* question types were few in sample size (only 12 and
3015 22 questions, respectively). The emotion proportion distribution of these question
3016 types are quite different to the others. Given the low number of questions, the sample
3017 is too small to make confident assessments. Furthermore, our assignment of Beyer
3018 et al.’s question type taxonomy was single-label; a multi-labelled approach may work
3019 better, however analysis of results would become more complex. A multi-labelled
3020 approach would be indicative for future work.

3021 6.6.2 External validity

3022 EmoTxt was trained on questions, answers and comments, however our dataset
3023 contained questions only. It is likely that our results may differ if we included other
3024 discussion items, however we wished to understand the emotion within developers’
3025 *questions* and classify the question based on the question classification framework
3026 by Beyer et al. [40]. Moreover, this study has only assessed frustrations within
3027 the context of a concrete domain; intelligent CVSs. The generalisability of this
3028 study to other IWSs, such as natural language processing services, or conventional
3029 web services, may be different. Furthermore, we only assessed four popular CVSs;
3030 expanding the dataset to include more services, including non-English ones, would
3031 be insightful. We leave this to future work.

3032 6.6.3 Construct validity

3033 Some posts extracted from SO were false positives. Whilst flagged for removal, we
3034 cannot guarantee that all false positives were removed. Furthermore, SO is known
3035 to have questions that are either poorly worded or poorly detailed, and developers
3036 sometimes ask questions without doing any preliminary investigation. This often
3037 results in down-voted questions. We did not remove such questions from our dataset,
3038 which may influence the measurement of our results.

3039 6.7 Conclusion

3040 We wanted to see how developers emotions are indicated in Stack Overflow (SO)
3041 posts when using CVSs. We analysed 1,425 SO posts about CVSs for emotions
3042 using an automated tool and then cross-checked our results manually. We found
3043 that the distribution of emotion differs across the taxonomy of issues, and that the
3044 current emotion model typically used in recent works is not appropriate for emotions
3045 expressed within SO questions. Consistent with prior work [216], our results
3046 demonstrate that ML classifiers for emotion are insufficient; human assessment is
3047 required.

CHAPTER 7

3048

3049

3050 Using Emotion Classification Models against Stack Overflow[†]

3051

3052 **Abstract** Pre-trained artificial intelligence (AI) models are increasingly available as APIs
3053 and tool-kits to software developers, making complex AI-enabled functionality available
3054 via standard and well-understood methods. However, reusing such models comes with
3055 risks relating to the lack of transparency of the model and training data bias, making it
3056 difficult to confidently employ the toolkit in a new situation. Vendors are responding and
3057 proposing artefacts such as model cards and datasheets to make models and their training
3058 more transparent. But is this enough? As part of an empirical investigation determining if
3059 a cloud-based intelligent web services (IWSs) was ready for production use, we processed
3060 developer questions on Stack Overflow using a published pre-trained classifier that was
3061 specifically tuned for the software engineering domain. In this paper, we present lessons
3062 learnt from this automation effort. We found unexpected results which led us to delve into
3063 model and training data—an option available to us because the information was available
3064 for research. We found that, had a model card and datasheet been prepared, we could
3065 have identified risks to our study much earlier on. However, model cards and datasheet
3066 specifications are not yet mature enough and additional tools and processes are still required
3067 to confirm a decision whether a trained model can be reused with confidence.

3068 7.1 Introduction

3069 Pre-trained artificial intelligence (AI) models are increasingly available to software
3070 developers either directly or wrapped into web-based components and toolkits; for
3071 example, Google’s Cloud AI¹ or Microsoft Azure’s Cognitive Services.² The grand

[†]This chapter is originally based on U. M. Graetsch, A. Cummaudo, M. K. Curumsing, R. Vasa, and J. Grundy, “Using Pre-Trained Emotion Classification Models against Stack Overflow Questions,” in *Proceedings of the 33rd International Conference on Advanced Information Systems Engineering*. Melbourne, VIC, Australia: Springer, 2021, In Review. Terminology has been updated to fit this thesis.

¹<https://bit.ly/2VheoH2> last accessed 29 Nov 2020.

²<https://bit.ly/37jiwvU> last accessed 29 Nov 2020.

promise is the rapid creation of AI-infused functionality into end-applications as developers can simply reuse models instead of training them from scratch, as training is laborious and resource-intensive [295]. Vendors do provide usage guidelines, component documentation, code examples and a compelling marketing narrative, although the limitations and risks are not as well-presented in official documentation [88, 91]. In practice, developers and technical architects study issue trackers and online forums such as Stack Overflow (SO) to assess and inform their decisions. Multiple studies highlight the value and insights to be gained from these online forums [2, 91, 338].

This work began as an investigation determining whether these services are production-ready for certain industry use cases (e.g., computer vision). Inspired by the possibility of finding insight from content in the online forums, we wanted to analyse the issues raised on SO by developers that relate to computer vision-based intelligent web services (IWSs), i.e., computer vision services (CVSs). Although manual analysis is feasible for this task, we decided to use a pre-trained natural language processing technique for a more automated approach to understand developers' frustration. This was motivated by (i) the gain from automation—specifically having an efficient, repeatable process and, more importantly, (ii) to learn about potential issues when using pre-trained models in a related, but new, contexts. Section 7.2 explains this in further detail.

In our analysis, beyond the direct summative aspects, we focused on emotions within the content posed on the online forums. This was motivated by work done by Wrobel [384], who suggested that some negative emotions can pose risk to developer productivity. However, while anger is a negative emotion, it can (in some people) generate a motivating response [384]. Our goal was to determine whether negative emotions (and specifically, *which* negative emotions) are the predominant theme within questions on these forums regarding IWSs. The natural expectation is that developers would not pose questions unless they needed support and help, and we expected to find a high prevalence of anger-based emotion in the questions (frustration that the service is not working as they think should), and perhaps surprise at any unexpected behaviour. Similarly, we would expect the tone of responses to questions to be neutral, and hopefully supportive. Our focus, however, remains on the questions posed.

Our findings, elaborated further in Section 7.4, were surprising. While the pre-trained model we selected was trained specifically on SO and tuned for emotions [67, 262], our results show that 14% issues were considered by the model with the positive emotions of *Love* or *Joy*, and only a surprisingly small amount (5%) fell into *Anger* (or frustration). A closer examination using multiple human reviewers showed an even more interesting insight: human reviewers did not agree with the automated machine classification, and worse, the reviewers did not agree with each other, suggesting that training machines with a consistent set of labels is a non-trivial exercise. Finally, we reflected whether the pre-trained classifier could be better documented. We found vendors are recognising these challenges and are offering solutions to better document their models [133, 245]. However, when we looked into the information captured by these solutions, we found their specification to be

3117 very broad and additional guidance for completion is required to help evaluate risks
3118 faced in an industry context (discussed in Section 7.5).

3119 7.2 Motivation

3120 The initial context of our work was to explore reusable cloud-based CVSs,³ arising
3121 for use in an industry context on a client project. Our prior research has identified
3122 growth in questions on SO relating to such services [91], thus enabling us to explore
3123 a rich dataset about developers' concerns about these pre-packaged and cloud-based
3124 AI-components.

3125 Aware that productivity of software developers can be adversely impacted by
3126 negative emotion [384], we decided to explore the emotions within our dataset
3127 expressed by developers through the questions they pose on SO. Our intent was
3128 to identify whether developers are surprised, angry, frustrated, or overall positive
3129 about using these CVSs (as expressed as emotions in their SO questions). This
3130 was motivated by prior work, which shows that—despite their technical nature—SO
3131 questions do exhibit emotion [67, 261]. Although we could have read these posts
3132 manually, for consistency, repeatability, and efficiency, we chose to automate this
3133 process by utilising an emotion-aware text classification system trained specifically
3134 on SO data [262]. Our expectation was that we would gain some insight into
3135 the questions through the emotions, and we hypothesised that we would see a high
3136 proportion of surprise (i.e., the API does not work as expected) and anger (frustration
3137 due to mismatched expectations).

3138 To permit replication, the raw results produced from this case study are made
3139 available online at <https://bit.ly/3eSp7ku>.

3140 7.3 Method

3141 We selected a classifier included in the EMTk toolkit that was specifically trained
3142 for emotional text classification in the software engineering domain [67]. The
3143 EMTk toolkit is available with a fully labelled training dataset [262], permitting
3144 reuse and analysis of internals. The classifier is based on Shaver et al.'s emotional
3145 hierarchy model [326] and performs binary classifications against text data provided
3146 in input files and an input parameter designating the emotion to be classified—one
3147 of *Love, Joy, Surprise, Fear, Sadness* or *Anger*. The classifier utilises support vector
3148 machine (SVM) classification and a Distributional Semantic Model (DSM) built
3149 using Word2Vec. This model is trained on 20 million SO posts. The DSM approach
3150 facilitates classification to take into consideration the surrounding context of the
3151 word, in addition to the polarity of individual words [68].

3152 As input for the classifier, we used a dataset of 1,425 SO questions restricted to
3153 intelligent CVSs, available from Cummaudo et al. [91]. We ran the classifier with
3154 the same input dataset for each of the six emotions. To cross-check classified output,
3155 we manually annotated a random sample of 300 questions with zero or more of the

3Such as Google Cloud Vision, Azure Computer Vision, or Amazon Rekognition

Table 7.1: Emotion classification frequencies.

Emotion	Frequency	Proportion
Love	103	7.2%
Joy	100	7.0%
Surprise	223	15.6%
Sadness	70	4.9%
Fear	224	15.7%
Anger	76	5.3%
No Emotion	622	43.6%

Table 7.2: Results from Inter-Rater Agreements.

Emotion	Three Raters	Three Raters + Classifier
Love	0.13 (<i>slight</i>)	0.19 (<i>slight</i>)
Joy	0.23 (<i>fair</i>)	0.13 (<i>slight</i>)
Surprise	0.15 (<i>slight</i>)	0.11 (<i>slight</i>)
Sadness	-0.01 (<i>poor</i>)	0.00 (<i>poor</i>)
Fear	0.25 (<i>fair</i>)	0.07 (<i>slight</i>)
Anger	0.05 (<i>slight</i>)	0.04 (<i>slight</i>)
No Emotion	0.09 (<i>slight</i>)	0.10 (<i>slight</i>)

³¹⁵⁶ six emotions. Each of these 300 posts were assigned to three raters who individually
³¹⁵⁷ carried out the following three steps: (i) identify the presence of emotion(s); (ii) if
³¹⁵⁸ emotion(s) exists, classify the emotion(s) under one or more of the six basic emotions
³¹⁵⁹ as per the Shaver framework. The coding guidelines provided by Novielli et al. [262]
³¹⁶⁰ were adhered to to assist with emotion classification per post. After collating each
³¹⁶¹ rater's results, we calculated a Fleiss' Kappa (κ) [118] as a measure of inter-rater
³¹⁶² agreement per emotion for each of the three human raters (manual rating), using
³¹⁶³ the `irr` computational R package [128] per suggestions provided in [149]. Once
³¹⁶⁴ completed, the three raters discussed discrepancies between posts classified with
³¹⁶⁵ different emotion where agreement for that emotion was low, however we did not
³¹⁶⁶ change any emotions that were initially assigned as this would impact reliability of
³¹⁶⁷ our interpretation of Novielli et al. [262]'s guidelines. We then used the results from
³¹⁶⁸ the classifier as a 'fourth' *automated* rater, comparing the results with the manual
³¹⁶⁹ rating by calculating the agreement for each emotion and Fleiss' Kappa for further
³¹⁷⁰ inter-rater agreement analysis.

³¹⁷¹ Of the 1,425 SO questions, the classifier did not classify any emotion in 622 posts
³¹⁷² (labelled *No Emotion*). The remaining posts were classified as: 224 posts as *Fear*,
³¹⁷³ 223 as *Surprise*, 70 as *Sadness*, 103 as *Love*, 100 as *Joy*, and 76 as *Anger*. Some posts
³¹⁷⁴ were classified against two or more emotions, and as a result, the total proportions
³¹⁷⁵ do not add up to exactly 100%. See Table 7.1. Results from our inter-rater analysis
³¹⁷⁶ are reported in Table 7.2.

³¹⁷⁷ Guidelines of indicative strengths of agreement are provided by Landis and Koch
³¹⁷⁸ [209], where: $\kappa \leq 0$ indicates *poor* agreement; $0 < \kappa \leq 0.2$ indicates *slight* agree-
³¹⁷⁹ ment; $0.2 < \kappa \leq 0.4$ indicates *fair* agreement; $0.4 < \kappa \leq 0.6$ indicates *moderate*

3180 agreement; $0.6 < \kappa \leq 0.8$ indicates *substantial* agreement. These interpretations
3181 suggest that, when using the classifier’s output as a fourth ‘rater’, there was slight
3182 agreement on all emotions except *Sadness*, where agreement was poor. Agreement
3183 amongst the three human raters was slight for *Love*, *Surprise*, *Anger* and *No Emotion*,
3184 fair for both *Joy* and *Fear*, and poor for *Sadness*.

3185 7.4 Results

3186 In this section, we present our findings with respect to limitations in the classifier
3187 and our investigation of the dataset that was used to train the classifier. Given the
3188 weak results, we then discuss whether model cards [245] and/or datasheets [133]
3189 could have provided a more effective approach to informing the viability and limits
3190 of the pre-trained model.

3191 7.4.1 Limitations of the Text Classifier

3192 The classifier did not assign any emotion to more than 43% of the SO posts. This
3193 result corroborates the findings by Murgia et al., who identified via a manual process
3194 *No Emotion* as the most prevalent classification [248]. For illustration, we provide
3195 a set of examples in Table 7.3. (The ratings column indicates the emotion labels
3196 assigned by each of the three human raters $R_{1..3}$ and the label assigned by the
3197 classifier C .) The first example given in Table 7.3 illustrates a neutral example,
3198 where none of the raters, including the classifier identified any emotion. In the
3199 second example, the classifier did not detect any emotion, however all three human
3200 raters agreed that the question indicated *Sadness*. In the third example, each rater
3201 identified different emotions, thereby indicating complete disagreement. In the
3202 fourth example, the classifier interpreted the question as *Joy*, whereas the human
3203 raters identified *Surprise* and *Anger*. Whilst that question had a word typically
3204 associated with *Joy* (i.e., “I’m pretty sure...”), the realistic context here is that the
3205 phrase ‘pretty’ indicates no emotion and the wider context of the question shows
3206 how the human raters identified a sense of frustration (anger) and surprise at the
3207 results the developer is finding. Lastly, two, additional examples are presented in
3208 the last and second-last rows of Table 7.3 to highlight different inconsistencies both
3209 between human raters and the classifier.

3210 We investigated our training dataset and related research documentation to see
3211 if that would give us further insights. We found two areas warranting further
3212 exploration—training data balance and training data annotation.

3213 7.4.2 Data imbalance

3214 We found that the purpose of the training dataset was actually to train two classifiers—
3215 a sentiment classifier and an emotional classifier. Each post in the training dataset
3216 was labelled with zero, one or more emotions. In addition, emotions were grouped,
3217 i.e., the positive emotions of *Joy* and *Love* were grouped into positive sentiment
3218 while *Sadness*, *Anger* and *Fear* were grouped into negative sentiment. *Surprise*

Table 7.3: Human Raters (R_1 , R_2 , R_3) versus automated classifier (C). Questions located at: [https://stackoverflow.com/q/\[ID\]](https://stackoverflow.com/q/[ID]).

Question ID and Quote	Ratings
[42375271] “Can we use Microsoft Emotion API in our Android Apps, considering the fact that it’s still in its ‘Preview’ mode...can we create our own customized app using the code of EMOTION API to recognize the moods of users in our own app?”	[C]: No Emotion [R_1]: No Emotion [R_2]: No Emotion [R_3]: No Emotion
[55599305] “I have consumed the google cloud vision api to recognize a document with a table, but sometimes the image will be a little rotated, im trying to get the value using theoef the key i want, but how do i get it if it’s not on the same.I was thinking of making a ‘line’ above and below the and finding if the point is between that, but i dont know how to do it.”	[C]: No Emotion [R_1]: Sadness [R_2]: Sadness [R_3]: Sadness
[43534783] “Can someone try Google VisionAPI FaceTracker and see if it works? ...All I get when I try running it is a black screen (after fixing). I don’t get any errors in the logs either.”	[C]: Fear [R_1]: Surprise [R_2]: No Emotion [R_3]: Anger
[51444352] “I’m pretty sure I set up my IAM role appropriately (I literally attached the ComprehendFullAccess policy to the role) and the Cognito Pool was also setup appropriately (I know this because I’m also using Rekognition and it works with the IAM Role and Cognito ID Pool I created) and yet every time I try to send a request to AWS Comprehend I get the error... Any idea of what I can do in this situation?”	[C]: Joy [R_1]: Surprise [R_2]: Surprise [R_3]: Anger
[50190527] “I am trying to perform OCR on pdf documents using google cloud vision API, i uploaded a pdf document into a cloud bucket and downloaded the oauth key file and added it in the script as below. But when i run the file, i get the permission denined: 403 error, can anyone please give me instructions on how to fix it, i did extensive google search and did not yield any results, i am surely missing something here... I have checked the older stack overflow questions and the links provided in answers are not active anymore.Thanks in advance for your help.”	[C]: No Emotion [R_1]: Sadness [R_2]: No Emotion [R_3]: Anger
[48145425] “I am Deploying Google cloud vision Ocr in My angular2 webapp. but i am getting many of the errors when i add this code in my webapp code. please help me to sort out this.”	[C]: Fear [R_1]: Fear [R_2]: No Emotion [R_3]: Sadness

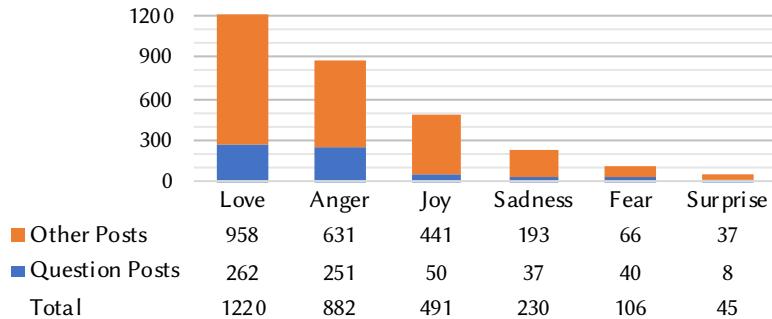


Figure 7.1: The emotion classifier training dataset distribution is largely skewed toward *Love*, resulting in data imbalance. (*No Emotion* labels were removed from this graph.)

3219 was assigned either positive or negative sentiment, depending on context [68, 262].
 3220 Figure 7.1 shows the distribution of emotion labels across 4,800 posts in the training
 3221 dataset; *No Emotion* ($n = 1959$) is removed to emphasise emotion-only results.
 3222 Note: some posts in the training data set were classified as more than 1 emotion,
 3223 hence the total counts add up to greater than 4800.

3224 Class imbalance and its impact on classifier models is a known problem in ma-
 3225 chine learning (ML) [223, 374], where one class (known as the majority, positive
 3226 class) significantly outnumbers the other class (known as the minority, negative
 3227 class). The impact of class imbalance on classification models results in minority
 3228 classes with lower precision and lower recall than the majority class, since the clas-
 3229 sifier does not generate rules for the minority class. One set of relevant techniques
 3230 for addressing class imbalance is data sampling; including undersampling or sub-
 3231 sampling, oversampling and hybrid approaches [223]. Whilst the training dataset
 3232 seems balanced for the purpose of sentiment analysis, there is a lack of balance
 3233 across individual emotions. The predominant emotion in the training dataset was
 3234 *No Emotion* at 40.8% of total posts. The most dominant emotions were *Love* and
 3235 *Anger* at 25% and 18% respectively. Less than 1% of posts were labelled with
 3236 *Surprise*. This means that the number of posts falling into some of the categories,
 3237 for example, *Surprise* and *Fear* (i.e., 45 and 106 posts, respectively) is very low for
 3238 training purposes.

3239 Further, the training dataset was spread across different types of SO posts (i.e.,
 3240 questions posts, answer posts, question comments, answer comments) to capture
 3241 the different emotional language, however our study was interested in classifying
 3242 SO question posts only. Of the training dataset's 4,800 posts, only 1,044 could
 3243 be identified as question posts and within that subset of posts, the distribution of
 3244 emotions was more polarised than in the overall 4,800 posts. *Love* and *Anger* are
 3245 the most predominant emotions in the training dataset, however *Anger* has a higher
 3246 proportion (24%) in question posts, as opposed to only 18.4% in the overall dataset.

3247 In summary, the training dataset was not balanced within each emotion category
 3248 and some emotions had very low sample numbers, as emphasised in the skew in
 3249 Figure 7.1. Proportions of training data examples per question per emotion was very
 3250 low for *Joy*, *Surprise*, *Sadness* and *Fear*. To address this imbalance and achieve

3251 better performance, training data could be enhanced to include additional samples
3252 or to use an oversampling approach. A recent study into class-balancing approaches
3253 in the context of defect prediction models found that class rebalancing does lead to
3254 a shift in the learned concepts [356].

3255 7.4.3 Emotion Labeling Bias

3256 In software engineering, hierarchical categorical emotional frameworks—including
3257 those featured in Parrott [274], Ekman et al. [110] and Shaver et al. [326]—have
3258 been assessed by researchers and pragmatically selected as the basis for training
3259 emotional classifiers. The chosen emotion framework is then used as the taxonomy
3260 of truth labels for classifier training datasets. Data for labeling is sourced from
3261 systems such as SO and JIRA [127, 248, 262, 269]. In the software engineering
3262 domain, truth labeling of emotions has, to date, been done manually [127, 248, 262].
3263 Emotion annotation involves at least a pair of annotators [12, 137]. For the EMTK
3264 training dataset, annotation was performed manually by a team of 12 coders, divided
3265 into four groups of three with a computer science background [67, 262]. Manual
3266 annotation challenges when coding emotions can be encountered due to different
3267 levels of semantic ambiguity within emotions and how humans express emotions in
3268 text [151].

3269 In the absence of an objective emotional truth, researchers' consistency is taken
3270 as a measure of correctness—i.e., multiple annotators that agree [248]. A measure
3271 of inter-rater agreement is Cohen's Kappa [81] (for two raters) or Fleiss' Kappa [118]
3272 for more raters. For the training dataset, inter-rater agreement ranged from $\kappa = 0.30$
3273 (fair) for *Joy* to $\kappa = 0.66$ (substantial) for *Love*. The researchers specifically trained
3274 dataset coders for consistency. The challenge of this approach with a subject such as
3275 emotions is the opportunity for bias. In contrast, in other studies that used annotation,
3276 researchers specifically attempted to reduce the opportunity for biases by including
3277 raters with different nationalities, skills, cultural backgrounds, by increasing the
3278 number of raters [269] and opting against consistency training [9]. As such, the
3279 approach taken to achieve consistency and makeup of label coders is important
3280 information for downstream consumers of an AI model.

3281 7.4.4 Emotion Labelling and Classification Granularity

3282 Training data annotation was performed on SO posts—which included questions,
3283 answers, and comments to questions and answers. Emotion annotation can be
3284 performed at different levels of granularity—word level [339], spans of words in a
3285 sentence [12], sentence level or larger. While a word level or keyword approach is
3286 considered too granular (as it does not capture the emotional context sufficiently),
3287 there is a risk of emotion progression during narratives and also within sentences [12,
3288 248]. Our CVS dataset consisted of questions only as we were seeking to assess
3289 developer emotion expressed at the time of raising the question. Question posts are
3290 typically longer than comments and may contain multiple emotions expressed at
3291 different levels of intensity that are interpreted differently by different readers.

3292 For example, see the fifth question in Table 7.3. We see that the first sentence
3293 does not carry any emotion, as the author is stating the steps to reproduce their
3294 issue. However, in the second sentence—where the API generates a “403 error”—
3295 the author expresses a mix of both *Sadness* and *Anger* (i.e., frustration) since their
3296 “extensive google search” yielded no results, for which they begin to self-doubt
3297 (“i am surely missing something here”). Lastly, *Love* is demonstrated in the last
3298 sentence, via appreciation in advance for potential responses to their question.

3299 7.5 Discussion

3300 There is a growing trend emerging from key industry vendors to better document pre-
3301 trained models using various means. For example, Google has proposed model cards
3302 to communicate performance characteristics of pre-trained models [245]. Google
3303 has also published sample model cards relating to their Cloud Vision API for face
3304 and object detection⁴ and, more recently, released a model card toolkit⁵ to encourage
3305 other ML practitioners to produce their own. Furthermore, this toolkit is now
3306 integrated into the Python library *scikit-learn* to help developers automatically
3307 generate model cards.⁶ Microsoft has focused on a standardised process of dataset
3308 documentation through datasheets to encourage transparency and accountability by
3309 documenting the motivation, composition, collection process and intended uses of
3310 data [133]. This is a key building block of the ‘Responsible ML’ initiative led by the
3311 partnership on AI,⁷ which aims to increase the transparency of AI and accountability
3312 of ML system documentation.⁸ Lastly, IBM too has proposed a ‘FactSheet’ concept
3313 combining model and data information [15].

3314 These tools are being adopted by organisations and researchers; for example,
3315 Open AI has published a basic model card of their generative language model⁹ and
3316 Google provided a sample model card for its toxicity analyser in its model card
3317 proposal paper [245]. Model cards are also being considered for high stakes envi-
3318 ronments such as clinical decision making [324], where they facilitate overarching
3319 governance regimes on how and when models can be used.

3320 In our case study, the combination of a model card for the classifier as well as a
3321 datasheet for training data would have provided valuable, easy to digest, and initial
3322 support to help evaluate whether the classifier is right for our context. However, the
3323 current specification of datasheet contents is very broad and lacks detailed directions
3324 for those completing required information. The model cards proposed by Google
3325 are focused on performance characteristics and do not sufficiently focus on the
3326 underlying data that was used to train and, hence, define the context of the classifier.

3327 Had all the required information been provided to sufficient detail, including
3328 a highlight of the importance of rater consistency training, we could have better

⁴<https://bit.ly/2IXDLe1> last accessed 28 November 2020.

⁵<https://bit.ly/3k7rLnk> last accessed 28 November 2020.

⁶<https://bit.ly/36bXnEK> last accessed 28 November 2020.

⁷<https://bit.ly/33kebYc> last accessed 28 November 2020.

⁸<https://bit.ly/31f8WPD> last accessed 28 November 2020.

⁹<https://bit.ly/3o6ECsj> last accessed 30 November 2020.

assessed risks and clarified at the outset whether an automated emotion assessment was an appropriate exercise. Further, with this information, we would be able perform our study with an more extensive rater consistency training, as well as a better appreciation of the limits of the classifier.

Hence, model cards and datasheets present rich opportunities for improving confidence and understanding in pre-trained AI technology. Now that toolkits are becoming increasingly available to make it easier for developers to generate toolkits, we suggest further research to evaluate model cards and datasheets (and combinations of the two) before pre-trained models are selected for specific tasks. This would make a valuable case study which we leave open for future work. Further, development of guidelines for model cards and datasheet creation, use and maintenance based on empirical evidence is also largely missing in literature; another avenue for potential research especially for use in industry contexts. A key challenge identified by this case study was the difficulty of validating results of an emotional classifier. An additional research study could aim to capture developer emotion directly as they log questions and facilitate learning of developer emotion classification through this direct method (e.g., a think-aloud study). This proposed approach to capturing data may shed further light into the emotional state an individual developer is experiencing *as they write* their questions. However, it would be of interest to assess if it is possible to draw conclusions about emotions that developers feel *in general*, due to the subjective nature of emotion. That is, it is possible that different developers would report a range of emotions even when they write similar posts.

Once validation of the results can be improved, additional improvement could be considered for the EMTk classifier including training it on questions only and using some of the identified data balancing techniques to re-balance the dataset. Another area of potential research is whether providing feedback to developers about the emotional content in their posts would change what they communicate. For instance, would it assist developer productivity if they were made aware of the emotional content of their contributions/posts?

7.6 Threats to Validity

This case study represents only one detailed example of a classifier trained on the emotional model proposed by Shaver et al. [326], documented in academic articles aimed to support research [67, 68, 261, 262]. This impacts the external validity of our study as the results cannot be generalised to other domains or emotion classifiers. To mitigate this, it would be very useful and informative to compare and validate our findings across a number of classifiers, however this is challenging since there is generally a lack of detailed information (i.e., model cards and datasheets) for available classifiers to support the analysis. This said, even a simple comparative analysis of emotion classification outputs is difficult because emotional classifiers are typically trained on a specific emotional model. A mapping between emotional models would therefore be needed, which demands expertise beyond software engineering research.

Another key limitation is that our analysis focused on SO questions on a particular

topic, whereas the EMTk had been trained on a mixture of different posts and topics.
3372 This again impacts the external validity of our results. It is not appropriate to draw
3373 a general conclusion from this analysis that emotions cannot be reliably classified
3374 by analysing text. In fact, there were higher inter rater scores achieved for EMTk's
3375 training dataset. Possibly additional rounds of clarification and moderation would
3376 yield a higher score and higher confidence.
3377

3378 It is common for questions on SO to be duplicates or downvoted, typically due
3379 to poor wording or a lack of detail in the body of the question. Duplicate and
3380 downvoted questions were not removed from our dataset used in the experiment,
3381 and, furthermore, any poorly worded questions may have impacted the automatic
3382 classifier's emotion labelling. This is likely to have impacted the measurement of
3383 our results.

3384 7.7 Related Work

3385 Emotion detection from text has been explored by researchers in depth. A recent
3386 survey of approaches, including the different emotion models and computational
3387 approaches, can be found in Sailunaz et al. [314] and Alswaidan and Menai [10].
3388 Recently, researchers have also explored deep learning, specifically bidirectional
3389 BLSTM models, to improve emotion detection from text [32]. Most approaches are
3390 supervised learning based, and hence rely on a labelled dataset for training.

3391 Some related work of special interest has been done in the area of sentiment
3392 analysis, where discussions touched on emotion recognition. Novielli et al. [261]
3393 investigated the suitability of using sentiment analysis tools to measure affect in SO
3394 questions and comments. In their analysis, they discussed that developers expressed
3395 negative emotions associated with their technical issues and that developers mainly
3396 express their frustrations for not being able to solve a problem. For questions with
3397 positive sentiment, they found that the positive lexicon did not express emotions, but
3398 rather positive opinions and use of positive speech acts associated with politeness
3399 and gratitude in advance of receiving a response. Also, of interest is the evaluation
3400 of sentiment analysis tools evaluated on SO, JIRA and App Review datasets by Lin
3401 et al. [216]. This study found that the prediction accuracy of the tools that were
3402 evaluated were biased against the majority class (neutral emotion).

3403 The use of biometric sensors is also an area of active research for software de-
3404 veloper emotion recognition. This includes conducting experiments with correlated
3405 sensor data analysing the emotions software developers present whilst working [139].
3406 Further work could include using the biometric-based data as a data source for truth
3407 labels for emotion analysis as developers write their questions on SO, supporting the
3408 proposed studies mentioned in Section 7.5.

3409 7.8 Conclusion

3410 We started this work with an idea to use existing AI techniques to *automatically* in-
3411 vestigate what other developers think of cloud IWSs. This translated into our attempt

³⁴¹² to use a pre-trained model that learnt from posts provided by software engineers on
³⁴¹³ SO. Developers learn, improve and deepen their skills from documentation, formal
³⁴¹⁴ or self-paced education, experience, and sharing their knowledge. Good documen-
³⁴¹⁵ tation often forms the foundation that enables learning and also to create educational
³⁴¹⁶ aids.

³⁴¹⁷ In this paper, we presented an observation case study that highlights a set of
³⁴¹⁸ gaps in how a peer-reviewed model, published in the field of software engineering,
³⁴¹⁹ lacks information about the limitations both within the documentation, as well as the
³⁴²⁰ articles published. To resolve these gaps, we investigated if new solutions that are
³⁴²¹ being proposed (such as model cards) would have been of use to us before conducting
³⁴²² our experiment. Model cards and datasheets will be a necessary and helpful first step,
³⁴²³ but as such we found their specification to be insufficient and additional guidance
³⁴²⁴ is required for those documenting the models cards and datasheets. Although we
³⁴²⁵ study only one pre-trained model in depth, our analysis shows that there are gaps
³⁴²⁶ in proposed solutions that can be addressed, and our future work will focus on
³⁴²⁷ investigating other models and IWSs to develop a more detailed documentation
³⁴²⁸ approach, specifically those that are being aimed for software engineering.

CHAPTER 8

3429

3430

3431

Better Documenting Computer Vision Services[†]

3432

3433 **Abstract** Using cloud-based computer vision services (CVSs) is gaining traction, where
3434 developers access AI-powered components through familiar RESTful APIs, not needing
3435 to orchestrate large training and inference infrastructures or curate/label training datasets.
3436 However, while these APIs *seem* familiar to use, their non-deterministic run-time behaviour
3437 and evolution is not adequately communicated to developers. Therefore, improving these
3438 services' API documentation is paramount—more extensive documentation facilitates the
3439 development process of intelligent software. In a prior study, we extracted 34 API docu-
3440 mentation artefacts from 21 seminal works, devising a taxonomy of five key requirements to
3441 produce quality API documentation. We extend this study in two ways. Firstly, by surveying
3442 104 developers of varying experience to understand what API documentation artefacts are
3443 of *most value* to practitioners. Secondly, identifying which of these highly-valued artefacts
3444 are or are not well-documented through a case study in the emerging CVS domain. We
3445 identify: (i) several gaps in the software engineering literature, where aspects of API docu-
3446 mentation understanding is/is not extensively investigated; and (ii) where industry vendors
3447 (in contrast) document artefacts to better serve their end-developers. We provide a set of
3448 recommendations to enhance intelligent software documentation for both vendors and the
3449 wider research community.

3450 8.1 Introduction

3451 Improving API documentation quality is a valuable task for any API. Succinct
3452 API documentation of good quality facilitates productivity [213, 251, 252], and
3453 therefore improved quality is better engineered into a system [237]. Where ap-
3454 plication developers integrate new services into their systems via APIs, their pro-

[†]This chapter is originally based on A. Cummaudo, R. Vasa, and J. Grundy, “Requirements of API Documentation: A Case Study into Computer Vision Services,” *IEEE Transactions on Software Engineering*, pp. 1–1, 2020, DOI 10.1109/TSE.2020.3047088. Terminology has been updated to fit this thesis.

ductivity is affected either by inadequate skills (“*I’ve never used an API like this, so must learn from scratch*”) or, where their skills are adequate, an imbalanced cognitive load that causes excessive context switching (“*I have the skills for this, but am confused or misunderstand*”). As a real-world use case, consider intelligent computer vision services (CVSs), in which an AI-based component produces a non-deterministic result based on a machine-learnt data-driven algorithm, rather than a predictable, rule-driven one [88]. These services use machine intelligence to make predictions on images such as object labelling or facial recognition [397, 408, 409, 410, 411, 418, 422, 430, 431, 432, 436, 450, 451, 484, 485]. The impacts of poor and incomplete documentation results in developer complaints on online discussion forums such as Stack Overflow [91]. Many comments show that developers do not think in the non-deterministic mental model of the designers who created the CVSs. They ask many varied questions from their peers to try and clarify their understanding.

It is therefore important to ensure developers have access to high-quality API documentation artefacts when consuming these services. Vendors should cover all documentation artefacts that the wider developer community find valuable, and the research community should aide in this process by investigating with types of information that comprise these artefacts, or the aspects of information design to best present this information. What causes a developer to be confused when using an API, and how to mitigate it via improved documentation, has been largely explored by researchers for *conventional APIs* (an overview is provided in Section 8.2). Various studies provide a myriad of recommendations into the value of API documentation artefacts based on both qualitative and quantitative analyses, involving developer opinions (from surveys), observation of developers, event logging or content analysis (see Figure 8.3). Such guidelines propose ways for developers, managers, and solution architects can construct systems better with improved documentation.

However, there does not yet exist a consolidated *systematic* review of this literature. Further, few studies offer a taxonomy to consolidate these guidelines together, and there still lacks a consolidated effort to capture guidelines on the requirements of good quality API documentation. Studies that produce these guidelines from literature are largely scattered across multiple sources. Investigating the ways by which these guidelines are produced can provide software engineering researchers with better insight into the research methods and data collection techniques used to produce these guidelines. Some studies, for example, use case studies, others use focus groups and brainstorming, or interviews and surveys. The extent to which researchers rely on developer opinion for API documentation guidelines is evident, and gaps in the methodological approaches that researchers use should be emphasised to shine light into new ways of conducting research in this important area. Furthermore, systematically capturing the information distilled from these guidelines into a readily accessible, consolidated taxonomy (designed to assist writing API documentation) must be validated in real-world circumstances to assess its efficacy with practitioners.

In our prior work, we proposed an API documentation taxonomy that was comprised of 21 key primary sources [87]. This paper significantly extends our previous

3500 work by addressing limitations in the existing taxonomy, thus refining it. Previously,
3501 we developed a metric for each dimension (topmost-layers) and category (leaf nodes)
3502 within the taxonomy [87]. This metric is an indication of the specific areas of API
3503 documentation software engineering researchers have focused their efforts, as mea-
3504 sured by the ratio of papers that investigated or reported various issues concerning
3505 the documentation artefacts defined within our taxonomy. For the context of this
3506 paper, we refer to this metric as an ‘in-literature’ score, or ILS. Within this paper,
3507 we build upon this facet but *in-practice* by assessing the efficacy of our taxonomy
3508 against developers using a survey instrument inspired by the System Usability Scale
3509 (SUS) [61]. Each artefact within the taxonomy is measured against this instrument
3510 for its utility, and a metric is produced to indicate how well developers *value* each
3511 of these artefacts. We refer to this metric as an ‘in-practice’ score, or IPS. (Details
3512 for how the IPS is calculated are given in Section 8.5.1.4.) We then identify the
3513 artefacts that are highly researched, the ones that developers demand the most, and
3514 where gaps in these artefacts remain for future research exploration.

3515 Lastly, while our prior work focused on *generalised* API documentation, in this
3516 extension, we apply our taxonomy to a case study of interest: i.e., better documenting
3517 CVSs. We empirically assess the taxonomy against three popular CVSs, namely
3518 Google Cloud Vision [422], Amazon Rekognition [397] and Azure Computer Vision
3519 [436]. For each category in our taxonomy, we assess whether the respective service’s
3520 documentation contains, partially-contains or does not contain the documentation
3521 artefact from our taxonomy, thus determining the extent to which the requirements
3522 of good API documentation are met within the vendors’ own documentation. From
3523 this, we triangulate each ILS and IPS value against the service’s level of inclusion
3524 of its respective documentation artefact, thereby making a judgement as to where
3525 the services can improve their documentation to make them more complete. Lastly,
3526 we present a ranking of each artefact for where research or vendors should be focus
3527 their documentation efforts that is of high value to both developers *and* to industry
3528 vendors.

3529 Thus, through this triangulation of the taxonomy with existing literature, utility
3530 to practitioners, and application via a case study (CVSs), we summarise three aspects
3531 of API documentation by identifying:

- 3532 (i) the documentation artefacts that been extensively studied by researchers, and
3533 those that warrant further attention by the software engineering research com-
3534 munity (via high/low ILS values);
- 3535 (ii) the documentation artefacts that are considered to be the most- and least-
3536 important from a practitioner’s point of view (via high/low IPS values);
- 3537 (iii) the documentation artefacts that have been well-established by vendors (via
3538 our case study on three prominent CVSs).

3539 To demonstrate how our taxonomy was developed, we include an extended
3540 revision of the systematic mapping study (SMS) from our existing work. The
3541 taxonomy we proposed consists of five key requirements: (1) Descriptions of API
3542 Usage; (2) Descriptions of Design Rationale; (3) Descriptions of Domain Concepts;
3543 (4) Existence of Support Artefacts; and (5) Overall Presentation of Documentation.

Following this, we developed a survey instrument to assess the overall utility of each of the artefacts that contribute towards these five requirements, which consisted of 43 questions of alternating positive and negative sentiment. We then narrow our focus down to our case study by applying the prioritised documentation artefacts (as identified by the survey) to three CVSs. Once our surveys were complete, we provide some general guidelines as to where cloud CVSs can make improvements to their API documentation. Lastly, we compare and contrast the results from our SMS to the results of the survey and of our case study, thereby identifying where future research efforts into API documentation should focus to give the biggest value back to practitioners.

Our key contributions in this work are:

- a score metric for each category that indicates where the highest research priorities have been in the existing literature;
- a score metric assessing the efficacy of the 34 categories that empirically reflects what artefacts are of the highest value from a *practitioner* point of view;
- a heuristic validation of each artefact against CVSs, assessing where existing CVS API documentation needs improvement;
- a number of practical recommendations for CVS vendors to better improve the quality of their API documentation; and
- an identification of the gaps for future research into API documentation based on the highest need by developers but, so far, has captured the least attention by researchers.

This paper is structured as follows: Section 8.2 presents related work; Section 8.3 is divided into two subsections, the first describing how primary sources were selected in the SMS with the second describing the development of our taxonomy from these sources; Section 8.4 presents the taxonomy; Section 8.5 describes how we developed a survey instrument of 43 questions to validate the taxonomy against developers, and assess its efficacy against the three popular CVSs selected; Section 8.6 presents the findings from our validation analysis; Section 8.7 describes the threats to validity of this work; and Section 8.8 provides concluding remarks and the future directions of this study. Additional materials are provided in Chapter C.

8.2 Related Work

8.2.1 Systematic Reviews in Software Documentation

Systematic reviews into how developers produce and use software documentation gives researchers consolidated insights into the efforts of multiple, disparate API documentation studies. For example, a recent 2018 study explored 36 API documentation generation tools and approaches, and analysed the tools developed and their inputs and documentation outputs [263]. The findings from this study emphasise that the largest effort in API documentation tooling is to assist developers

3584 to generate either example code snippets and/or templates or natural language de-
3585 scriptions of the API directly from the program’s source code. These snippets or
3586 descriptions can then be placed in the API documentation, thereby increasing the
3587 efficiency at which API documentation can be written. Additionally, tools from 12
3588 studies target the maintainability of existing APIs of existing APIs, while tools from
3589 11 studies target the correctness and accuracy of the documentation by validating
3590 that what is written in the documentation is accurate to the technical structure of
3591 the API. From the end-developer’s perspective, some tools (17 studies) help target
3592 improvements to the developer’s understandability and learnability of new APIs by
3593 linking in examples directly with questions such as on Stack Overflow. However,
3594 the results from this study regards the *tooling* used to either assist in producing,
3595 validating or learning from API documentation. While this is a systematic study
3596 with key insights into the types of tooling produced, there is still a gap for an SMS in
3597 what *guidelines* have been produced by the literature in developing natural language
3598 documentation itself—and how well developers *agree* to those guidelines—which
3599 our work has addressed.

3600 An extensive SMS into studies presented in the *overall* software documentation
3601 domain was given in Zhi et al. [391]. This study reviewed a set of 69 papers from
3602 1971 to 2011 to develop a systematic map on the various research aspects relating
3603 to documentation cost, benefit and quality, finding that 38% of papers propose novel
3604 techniques while 29% contribute empirical evidence (i.e., validation and evaluation
3605 papers—see Section 8.3.1.4). The authors find that a majority of papers discuss qual-
3606 ity aspects of software documentation, namely the quality attributes of completeness,
3607 consistency and accessibility, and that the main usage of software documentation re-
3608 gards maintenance aid and program comprehension. Another key insight—relevant
3609 to our study—found that, on average, survey-based studies into documentation in-
3610 volved 106 participants and generally these participants were from the same (or only
3611 two) organisations. However, unlike our study, this study formalises the documenta-
3612 tion efforts of *any* software document, and not exclusively into API documentation
3613 artefacts required to help developers produce software. Further, our study differs in
3614 that the results from our study are consolidated into a structured taxonomy, instead
3615 of a meta-model which Zhi et al. perform, which is then triangulated against a
3616 real-world use case (i.e., intelligent CVSs) and software developers via a survey.

3617 **8.2.2 API Usability and Documentation Knowledge**

3618 API usability and its impact on documentation knowledge is an imperative area of
3619 study, since it provides useful links between API documentation and more technical
3620 issues related to API design or tools. Extensive discussions from Myers and Stylos
3621 [252] and Myers et al. [251] encapsulate a 30-year effort to evaluate and improve
3622 API usability through lenses adapted human-computer interaction research. Es-
3623 sentially, by treating a developer as the ‘end-user’ of an API (i.e., interacting and
3624 programming with the API in their own systems), the authors discuss various case
3625 studies by which API usability was improved by various human-centred approaches,
3626 resulting in improved learnability of the API in addition to improved productivity

3627 and effectiveness in using the API. While the methods are primarily used for end-user
3628 usability testing, their observations highlight the importance of good aesthetic and
3629 interaction design of developer’s tooling and the need for new tooling to augment
3630 what developers already do to reduce learning overhead. An extensive review of the
3631 usability methods used, and their benefits to API usability, demonstrates how various
3632 techniques—grounded through established usability guidelines and frameworks—
3633 can be used to assess how an API’s usability impacts its key stakeholders (i.e., API
3634 designers, developers, and end-users). The role of API *documentation* in context to
3635 an API’s overall usability is imperative; for instance, limited documentation on a par-
3636 ticular API (and limited code snippets) is often a key complaint to poor API usability
3637 [252]. Exploring aspects on information design elements within API documentation
3638 is therefore critical to mitigate such complaints.

3639 In Watson [371], the authors performed a heuristic assessment from 35 popular
3640 APIs against 11 high-level universal design elements of API documentation. Of
3641 these 35 APIs, 28 were open-source software repositories and seven came from
3642 commercial independent software vendors. Two coders manually inspected each
3643 API’s respective documentation sets, starting from the documentation’s entry page
3644 and using the navigation features of the documentation to further explore the doc-
3645 umentation. Both coders evaluated each of the 11 heuristics, noting whether they
3646 could be found. This study highlighted how many APIs, even popular ones, fail
3647 to grasp these basic design elements. For example, 25% of the documentation sets
3648 did not provide any basic overview documentation to the API. Therefore, from a
3649 practitioner’s perspective, the study describes a high-level overview of how certain
3650 documentation artefacts address their needs and whether they are typically found in
3651 documentation. However, while the methodological approach used in this study to
3652 assess the heuristics is similar to our approach, the heuristics themselves used within
3653 Watson’s study is based on only three seminal works and only contains 11 design
3654 elements. Our study extends these heuristics and structures them into a consolidated,
3655 hierarchical taxonomy which we then validate against practitioners.

3656 A taxonomy of distinct knowledge patterns within reference documentation by
3657 Maalej and Robillard [226] classified 12 distinct knowledge types. Unlike our work,
3658 which uses an SMS of existing studies as the source of our taxonomy development,
3659 this study uses a grounded method via theoretical sampling of the API documentation
3660 of two mature (extensively documented) open source systems. This was performed
3661 by each author to elicit a list of knowledge types over an iterative six month process.
3662 The taxonomy was then evaluated against the JDK 6 and .NET 4.0 frameworks
3663 using a sample of 5574 documentation units and 17 trained coders to assign each
3664 knowledge type to the documentation unit. Results showed that the functionality
3665 and structure of these APIs are well-communicated, although core concepts and
3666 rationale about the API are quite rarer to see. The authors also identified low-
3667 value ‘non-information’—described as documentation that provides uninformative
3668 boilerplate text with no insight into the API at all—which was substantially present in
3669 the documentation of methods and fields in the two frameworks. They recommend
3670 that developers factor their 12 distinct knowledge types into the process of code
3671 documentation, thereby preventing low-value non-information, and thus developers

3672 can use the patterns of knowledge to evaluate the content, organisation, and utility
3673 of their own documentation. The development of their taxonomy consisted of
3674 questions to model knowledge and information, thereby capturing the reason about
3675 disparate information units independent to context; a key difference to this paper
3676 is the *systematic* taxonomy approach utilised and the source of information of our
3677 taxonomy (i.e., existing literature).

3678 8.2.3 Computer Vision Services

3679 Recent studies into cloud-based CVSs have demonstrated that poor reliability and
3680 robustness in computer vision can ‘leak’ into end-applications if such aspects are
3681 not sufficiently appreciated by developers. A study by Hosseini et al. [162] showed
3682 that Google Cloud Vision’s labelling fails when as little as 10% noise is added to the
3683 image. Facial recognition classifiers are easily confused by modifying pixels of a face
3684 and using transfer learning to adapt one person’s face into another [366]. Our own
3685 prior work found that the non-deterministic evolution of these types of services is not
3686 adequately communicated to developers [88], resulting in lost developer productivity
3687 whereby developers ask fundamental questions about the concepts behind these
3688 services, how they work, and where better documentation can be found [91]. This
3689 paper continues this line of research by providing a means for service providers to
3690 better document their services using a taxonomy and suggested improvements.

3691 8.3 Taxonomy Development

3692 We developed our taxonomy under two primary phases. First, we conducted an
3693 SMS identifying API documentation studies, following guidelines by Kitchenham
3694 and Charters [194] and Petersen et al. [282] (Section 8.3.1). A high level overview
3695 of this first phase is given in Figure 8.2. Second, we followed a software engineering
3696 taxonomy development method by Usman et al. [360] (Section 8.3.2) based on the
3697 findings of our SMS, which involved an extensive validation involving real-world
3698 developers and contextualised with computer vision APIs (Section 8.5).

3699 8.3.1 Systematic Mapping Study

3700 8.3.1.1 Research Questions (RQs)

3701 The first step in producing our SMS was to pose two RQs:

- 3702 • **RQ1:** What documentation ‘knowledge’ do API documentation studies con-
3703 tribute?
- 3704 • **RQ2:** How is API documentation studied?

3705 Our intent behind RQ1 was to collect as many studies provided by literature on
3706 how API documentation should be written using natural language, i.e., not using
3707 assistive tooling. In this regard, documentation ‘knowledge’ encompasses any nat-
3708 ural language API documentation artefact associated with the implementation of
3709 an application using a third-party API. As the goals of this study are to arrive at a

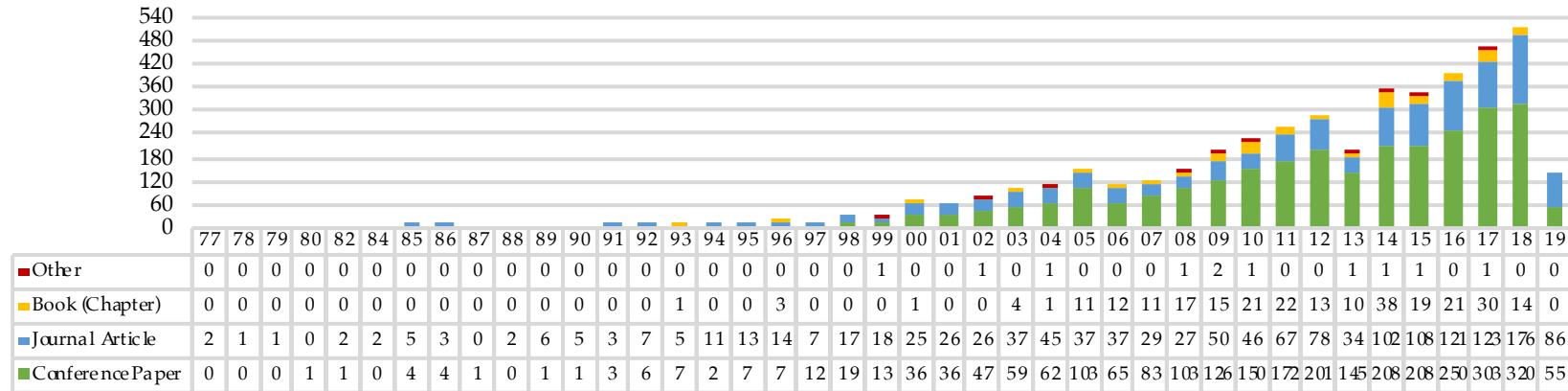


Figure 8.1: Search results by year and venue type.

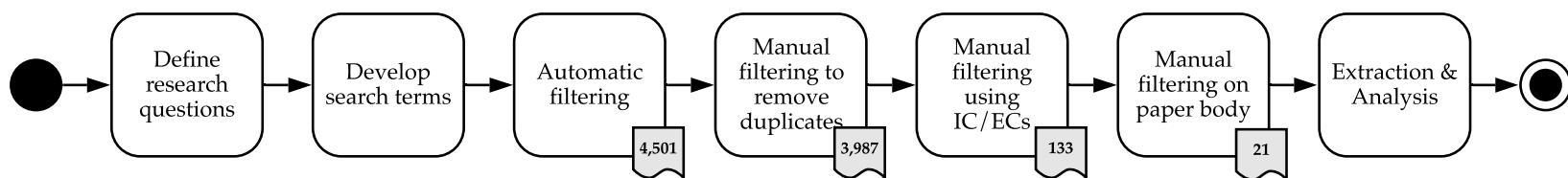


Figure 8.2: A high level overview of the filtering steps from defining and executing our search query to the data extraction of our primary studies. Number of accepted papers resulting from each filtering step is shown.

3710 taxonomy encapsulating the requirements of good API documentation (Section 8.4),
3711 we sought to arrive at studies that provide useful information to developers that
3712 informs the relevance and value of which aspects of API documentation are more
3713 useful than others. This captures the knowledge that developers need to know about
3714 what aspects of their APIs should be documented and the artefacts by which they do
3715 this. This helped us shape and form the taxonomy provided in Section 8.4. Secondly,
3716 RQ2's intent was to understand how the studies derive at their conclusions, thereby
3717 helping us identify gaps in literature where future studies can potentially focus.

3718 *8.3.1.2 Automatic Filtering*

3719 As done in similar software engineering studies [131, 140, 360], we explored auto-
3720 matic filtering of online databases. We defined which SWEBOK knowledge areas
3721 [167] were relevant to devise a search query. Our search query was built using re-
3722 lated knowledge areas, relevant synonyms, and the term 'software engineering' (for
3723 comprehensiveness) all joined with the OR operator. Due to the lack of a standard
3724 definition of an API, we include the terms: 'API' and its expanded term; software
3725 library, component and framework; and lastly SDK and its expanded term. These
3726 too were joined with the OR operator, appended with an AND. Lastly, the term
3727 '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)

3728 We executed the query on all available metadata (title, abstract and keywords) in
3729 May 2019 against Web of Science¹ (WoS), Compendex/Inspec² (C/I) and Scopus³.
3730 We selected three particular primary sources given their relevance in software en-
3731 gineering literature (containing the IEEE, ACM, Springer and Elsevier databases)
3732 and their ability to support advanced queries [60, 194]. A total 4,501 results⁴ were
3733 found, with 549 being duplicates. Table 8.1 displays our results in further detail (du-
3734 plicates not omitted); Figure 8.1 shows an exponential trend of API documentation
3735 publications produced within the last two decades. (As this search was conducted
3736 in May 2019, results taper in 2019.)

3737 *8.3.1.3 Manual Filtering*

3738 A follow-up manual filtering stage followed the 4,501 results obtained by automatic
3739 filtering. As described below, we applied the following inclusion criteria (IC) and
3740 exclusion criteria (EC) to each result:

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>.

Table 8.1: Search results and publication types

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

- ³⁷⁴¹ **IC1** Studies must be relevant to API documentation: specifically, we exclude
³⁷⁴² studies that deal with improving the technical API usability (e.g., improved
³⁷⁴³ usage patterns);
³⁷⁴⁴ **IC2** Studies must discuss artefacts that document APIs;
³⁷⁴⁵ **IC3** Studies must be relevant to software engineering as defined in SWEBOK;
³⁷⁴⁶ **EC1** Studies where full-text is not accessible through standard institutional databases;
³⁷⁴⁷ **EC2** Studies that do not propose or extend how to improve the official, natural
³⁷⁴⁸ language documentation of an API;
³⁷⁴⁹ **EC3** Studies proposing a third-party tool to enhance existing documentation or
³⁷⁵⁰ generate new documentation using data mining (i.e., not proposing strategies
³⁷⁵¹ to improve official documentation);
³⁷⁵² **EC4** Studies not written in English;
³⁷⁵³ **EC5** Studies not peer-reviewed.

³⁷⁵⁴ Each of these ICs and ECs were applied to every paper after exporting all
³⁷⁵⁵ metadata of our results to a spreadsheet. The first author then curated the publications
³⁷⁵⁶ using the following revision process.

³⁷⁵⁷ Firstly, we read the publication source—to rapidly omit non-software engineering
³⁷⁵⁸ papers—as well as the author keywords, title, and abstract of all 4,501 studies.
³⁷⁵⁹ As some studies were duplicated between our three primary sources, we needed to
³⁷⁶⁰ remove any repetitions. We sorted and reviewed any duplicate DOIs and fuzzy-
³⁷⁶¹ matched all very similar titles (i.e., changes due to punctuation between primary
³⁷⁶² sources), thereby retaining only one copy of the paper from a single database. Sim-
³⁷⁶³ilarly, as there was no limit to our date ranges, some studies were republished in
³⁷⁶⁴ various venues (i.e., same title but different DOIs). These were also removed using
³⁷⁶⁵ fuzzy-matching on the title, and the first instance of the paper’s publication was
³⁷⁶⁶ retained. This second phase resulted in 3,987 papers.

³⁷⁶⁷ Secondly, we applied our inclusion and exclusion criteria to each of the 3,987
³⁷⁶⁸ papers by reading the abstract. Where there was any doubt in applying the criteria
³⁷⁶⁹ to the abstract alone, we automatically shortlisted the study. We rejected 427 studies
³⁷⁷⁰ that were unrelated to software engineering, 3,235 were not directly related to docu-
³⁷⁷¹menting APIs (e.g., to enhance coding techniques that improve the overall developer
³⁷⁷² usability of the API), 182 proposed new tools to enhance API documentation or
³⁷⁷³ used machine learning to mine developer’s discussion of APIs, and 10 were not in
³⁷⁷⁴ English. This resulted in 133 studies being shortlisted to the final phase.

3775 Thirdly, we re-evaluated each shortlisted paper by re-reading the abstract, the
3776 introduction and conclusion. We removed a further 64 studies that were on API
3777 usability or non API-related documentation (i.e., code commenting). At this stage,
3778 we decided to refine our exclusion criteria to better match the research goals of
3779 this study by including the word ‘natural language’ documentation in EC2. This
3780 removed studies where the focus was to improve technical documentation of APIs
3781 such as data types and communication schemas. Additionally, we removed 26
3782 studies as they were related to introducing new tools (EC3), 3 were focused on tools
3783 to mine API documentation, 7 studies where no guidelines were provided, 2 further
3784 duplicate studies, and a further 10 studies where the full text was not available,
3785 not peer reviewed or in English. Books are commonly not peer-reviewed (EC5),
3786 however no books were shortlisted within these results. This final stage resulted in
3787 21 primary studies for further analysis, and the mapping of primary study identifiers
3788 to references S1–21 can be found in Appendix C.3.

3789 As a final phase, we conducted reliability analysis of our shortlisting method.
3790 We conducted intra-rater reliability of our 133 shortlisted papers using the test-
3791 retest approach suggested by Kitchenham and Charters [194]. We re-evaluated a
3792 random sample of 10% of the 133 shortlisted papers a week after initial studies were
3793 shortlisted. This resulted in *substantial agreement* [209], measured using Cohen’s
3794 kappa ($\kappa = 0.7547$).

3795 8.3.1.4 Data Extraction & Systematic Mapping

3796 Of the 21 primary studies, we conducted abstract key-wording adhering to Petersen
3797 et al.’s guidelines [282] to develop a classification scheme. An initial set of key-
3798 words were applied for each paper in terms of their methodologies and research
3799 approaches (RQ2), based on an existing classification schema used in the require-
3800 ments engineering field by Wieringa et al. [378]. These are: *evaluation papers*,
3801 which evaluates existing techniques currently used in-practice; *validation papers*,
3802 which investigates proposed techniques not yet implemented in-practice; *experi-
3803 ence papers*, which are written by practitioners in the field and provide insight into
3804 their experiences of adopting existing techniques; and *philosophical papers*, which
3805 presents new conceptual frameworks that describes a language by which we can
3806 describes our observations of existing or new techniques, thereby implying a new
3807 viewpoint for understanding phenomena. For example, documenting APIs using
3808 code snippets is a commonly used practice by developers (see the primary sources
3809 listed in Appendix C.1), and conducting an experiment exploring how quickly prac-
3810 titioners achieve this would be an evaluation paper. In contrast, a validation paper
3811 explores novel techniques that are proposed but not yet implemented in practice;
3812 for example, a paper proposing that APIs should document success stories so that
3813 developers know where, why, and how the API was successfully implemented may
3814 test this novel technique via field study experiments (e.g., interviewing developers
3815 on the new technique) without reference to real-world examples. A paper written
3816 by a group of developers sharing their insights into the improvements of their doc-
3817 umentation before and after providing extensive tutorials would be an experience

Table 8.2: Data extraction form

Data item(s)	Description
Citation metadata	Title, author(s), years, publication venue, publication type
Artefact(s) discussed	As per IC2, the study must identify at least one API documentation artefact
Evaluation method	Did the authors evaluate their proposed artefacts? If so, how?
Primary technique	The primary technique used to devise the artefact(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

³⁸¹⁸ paper. Philosophical papers may propose entirely new vocabulary to explore API
³⁸¹⁹ documentation, devising new frameworks from which other researchers can explore
³⁸²⁰ the field from a new viewpoint.

³⁸²¹ After all primary studies had been assigned keywords, we noticed that all papers
³⁸²² used field study techniques, and thus we consolidated these keywords using Singer
³⁸²³ et al.'s framework of software engineering field study techniques [331]. Singer et al.
³⁸²⁴ captures both study techniques *and* methods to collect data within the one framework,
³⁸²⁵ namely: *direct techniques*, including brainstorming and focus groups, interviews and
³⁸²⁶ questionnaires, conceptual modelling, work diaries, think-aloud sessions, shadowing
³⁸²⁷ and observation, participant observation; *indirect techniques*, including instrumenting
³⁸²⁸ systems, fly-on-the-wall; and *independent techniques*, including analysis of work
³⁸²⁹ databases, tool use logs, documentation analysis, and static and dynamic analysis.

³⁸³⁰ Table 8.2 describes our data extraction form, which was used to collect relevant
³⁸³¹ data from each paper. Figure 8.3 presents our systematic mapping, where each
³⁸³² study is mapped to one (or more, if applicable) of methodologies plotted against
³⁸³³ Wieringa et al.'s research approaches. We find that a majority of these studies
³⁸³⁴ survey developers using direct techniques (i.e., interviews and questionnaires) and
³⁸³⁵ some performing structured documentation analysis. Few studies report recent
³⁸³⁶ experiences; literature reports the artefacts that document APIs from evaluation
³⁸³⁷ research, in addition to some validation studies. There are few experience papers
³⁸³⁸ describing anecdotal evidence, and almost no philosophical papers that describe new
³⁸³⁹ conceptual ways at approaching API documentation as a large majority of existing
³⁸⁴⁰ work either evaluates existing (in-practice) strategies or validates the effectiveness
³⁸⁴¹ of new strategies.

³⁸⁴² 8.3.2 Development of the Taxonomy

³⁸⁴³ A majority of taxonomies produced in software engineering studies are often made
³⁸⁴⁴ extemporaneously [360]. For this reason, we decided to proceed with a systematic
³⁸⁴⁵ approach to develop our taxonomy using the guidelines provided by Usman et al.
³⁸⁴⁶ [360], which are extended from lessons learned in more mature domains. In this
³⁸⁴⁷ subsection, we outline the 4 phases and 13 steps taken to develop our taxonomy
³⁸⁴⁸ based on Usman et al.'s technique. Usman et al.'s final *validation* phase is largely
³⁸⁴⁹ detailed within Section 8.5 after we present our taxonomy in Section 8.4.

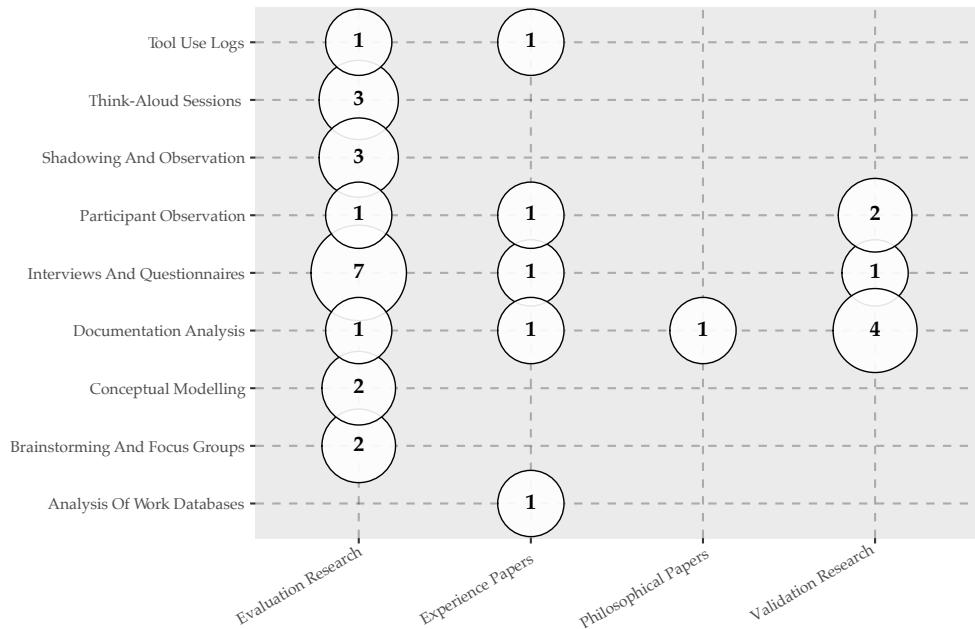


Figure 8.3: Systematic map: field study technique vs research type

3850 Formally, Usman et al. provides guidelines to define these units under the first
 3851 six stages under the planning phase. In our study, our preliminary phase involves
 3852 answering the following:

- 3853 (1) *define the software engineering knowledge area*: The software engineering
 3854 knowledge area, as defined by the SWEBOK, is software construction;
- 3855 (2) *define the objective*: The main objective of the proposed taxonomy is to define a
 3856 set of categories that enables to classify different facets of natural language API
 3857 documentation artefacts (not API *usability*) as reported in existing literature;
- 3858 (3) *define the subject matter*: The subject matter of our proposed taxonomy is
 3859 documentation artefacts of APIs;
- 3860 (4) *define the classification structure*: The classification structure of our proposed
 3861 taxony is *hierarchical*;
- 3862 (5) *define the classification procedure*: The procedure used to classify the docu-
 3863 mentation artefacts is qualitative;
- 3864 (6) *define the data sources*: The basis of the taxonomy is derived from field study
 3865 techniques (see Section 8.3.1.4).

3866 8.3.2.1 *Identification and extraction phase*

3867 The second phase of the taxonomy development involves (7) *extracting all terms*
 3868 *and concepts* from relevant literature, which we have achieved from our SMS. These
 3869 terms are then consolidated by (8) *performing terminology control*, as some terms
 3870 may refer to different concepts and vice-versa. For example, Watson defines one of
 3871 the heuristics used in the study's experiment as “sample apps to understand how to

use the elements of an API in context and as another source from which to copy program code... a sample app is a complete application that includes examples of the API as well as the other functions that comprise a complete program” [371]. In this case, the term ‘sample app’, ‘program code’, and ‘complete application’ were extracted as a term of interest and noted. Similarly, in Robillard [304], the phrase ‘applications’ is used to define a category of example code snippets which “consists of code segments from complete applications” and is generally some form of “demonstration samples sometimes distributed with an API... that developers can download from various source code repositories” [304]. Again, the phrase ‘complete applications’, ‘demonstration samples’, ‘download’, and ‘source code’ was identified as a terms of interest and noted. Once all papers were read, we consolidated a list of all of these noted highlights to help consolidate the terms and perform terminology control. In this example, the phrase ‘Downloadable source code demonstrating complete sample applications’ was consolidated from both Watson and Robillard’s studies, which—in addition to the other primary studies that iteratively changed wording slightly due to steps (9–10)—formed the basis of the taxonomy dimension [A7].

3889 8.3.2.2 *Design phase*

3890 The design phase identified the core dimensions and categories within the extracted
3891 data items. The first step is to (9) *identify and define taxonomy dimensions*; for this
3892 study we utilised a bottom-up approach to identify each dimension, i.e., extracting the
3893 categories first and then nominating which dimensions these categories fit into using
3894 an iterative approach. As we used a bottom-up approach, step (9) also encompassed
3895 the second stage of the design phase, which is to (10) *identify and describe the*
3896 *categories of each dimension*. Thirdly, we (11) *identify and describe relationships*
3897 between dimensions and categories, which can be skipped if the relationships are
3898 too close together, as is the case of our grouping technique which allows for new
3899 dimensions and categories to be added. The last step in this phase is to (12) *define*
3900 *guidelines for using and updating the taxonomy*. The taxonomy is as simple as a
3901 checklist that can be heuristically applied to API documentation, and each dimension
3902 is malleable and covers a broad spectrum of artefacts; while we do not anticipate
3903 any further dimensions to be added, new categories can easily be fitted into one of
3904 the dimensions (see Section 8.8). We provide guidelines for use in our application
3905 of the taxonomy against CVSSs within Sections 8.4 and 8.6.

3906 8.3.2.3 *Validation phase*

3907 In the final phase of taxonomy development, taxonomy designers must (13) *validate*
3908 *the taxonomy* to assess its usefulness. Usman et al. [360] describe three approaches to
3909 validate taxonomies: (i) orthogonal demonstration, in which the taxonomy’s orthog-
3910 onality is demonstrated against the dimensions and categories, (ii) benchmarking
3911 the taxonomy against similar classification schemes, or (iii) utility demonstration by
3912 applying the taxonomy heuristically against subject-matter examples. In our study,
3913 we adopt utility demonstration by use of a survey and heuristic application of the

3914 taxonomy against real-world case-studies (i.e., within the domain of CVSSs). This is
3915 is discussed in greater detail within Section 8.5.

3916 8.4 A Taxonomy for API Documentation

3917 Our taxonomy consists of five dimensions (labelled A–E). These five dimensions
3918 are made of 34 categories, which represent API documentation artefacts that con-
3919 tribute towards these dimensions. In the context of our taxonomy, a category can
3920 represent (i) discrete and self-contained documentation artefacts (e.g., quick start
3921 guides [A1]), (ii) additional information used to describe the API (e.g., licensing
3922 information about the API [D6]), or (iii) aspects regarding the information design of
3923 this documentation (e.g., consistent look and feel [E6]). Collectively, the categories
3924 form the *requirements* of good quality API documentation, as expressed through the
3925 five dimensions. When worded as questions, each dimension respectively covers the
3926 following:

- 3927 • **[A] Descriptions of API Usage:** *how* does the developer use this API for their
3928 intended use case?
- 3929 • **[B] Descriptions of Design Rationale:** *when* should the developer choose
3930 this particular API for their intended use case?
- 3931 • **[C] Descriptions of Domain Concepts:** *why* does the developer select this
3932 particular API for their application’s domain and does the API’s domain align
3933 with the application’s domain?
- 3934 • **[D] Existence of Support Artefacts:** *what* additional API documentation can
3935 the developer find to aid their productivity?
- 3936 • **[E] Overall Presentation of Documentation:** is the *visualisation* of the above
3937 information well organised and easy for the developer to digest?

3938 Further descriptions of the categories encompassing each dimension are given within
3939 Figure 8.4 and Appendix C.1, coded as $[Xi]$, where i is the category identifier within
3940 a dimension, $X \in \{A, B, C, D, E\}$.

3941 Appendix C.1 shows which of the primary sources (S1–21) reports aspects of
3942 the artefacts described as an ‘in-literature score’ (ILS). This score is calculated as
3943 a percentage of the number of primary studies that investigated or reported various
3944 issues regarding the specific artefact divided by the total of primary studies (see
3945 Section 8.6.1.2). This score is contrasted to the ‘in-practice score’ (IPS) which
3946 indicates the overall level of agreement that *practitioners* think such documentation
3947 artefacts are needed (see Section 8.6.1.1). For comparative purposes, we illustrate a
3948 colour scale (from red to green) to indicate the relevancy weight between ILS and IPS
3949 values in Appendix C.1 as per their assigned, discretised intervals (see Table 8.3). We
3950 also show illustrative interpretations of these generalised artefacts through italicised
3951 examples within Appendix C.1. We then provide three columns that assesses the
3952 presence of these documentation artefacts against three popular CVSSs: Google Cloud
3953 Vision, AWS’s Rekognition, and Azure Cloud Vision (abbreviated to GCV, AWS
3954 and ACV). A fully shaded circle (●) indicates that the documentation artefact was

- [A] Descriptions of API Usage
 - [A1] Quick-start guides
 - [A2] Low-level reference manual ✓
 - [A3] Explanation of high level architecture ✓
 - [A4] Introspection source code comments ✗
 - [A5] Code snippets of basic component function ✓
 - [A6] Step-by-step tutorials with multiple components
 - [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
 - [A12] Error definitions describing how to address problems

- [B] Descriptions of the API's Design Rationale
 - [B1] Entry-point purpose of the API ✓
 - [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

- [C] Descriptions of Domain Concepts behind the API
 - [C1] Relationship between API components and domain concepts
 - [C2] Definitions of domain terminology
 - [C3] Documentation for nontechnical audiences ✓

- [D] Existence of Support Artefacts
 - [D1] FAQs ✓
 - [D2] Troubleshooting hints ✗
 - [D3] API diagrams
 - [D4] Contact for technical support ✓
 - [D5] Printed guide
 - [D6] Licensing information

- [E] Overall Presentation of Documentation
 - [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 ✓

Figure 8.4: Our proposed taxonomy: The requirements of good-quality API documentation (dimensions) represented through individual documentation artefacts (categories).

3955 clearly found in the service, while a half-shaded circle (●) indicates that the artefact
3956 was only partially present. An outlined circle (○) indicates that the service lacks the
3957 indicated documentation artefact within our taxonomy. This empirical assessment
3958 is further detailed in Section 8.6.3, which outlines concrete areas in the respective
3959 services’ documentation where improvements could be made, as well as hyperlinks
3960 to the documentation where relevant.

3961 Figure 8.4 illustrates a condensed version of taxonomy. We provide iconography
3962 for the presence (●) or non-presence (○) of these artefacts in *all three* CVSSs assessed,
3963 per Section 8.6.1.1.

3964 8.5 Validating the Taxonomy

3965 8.5.1 Survey Study

3966 8.5.1.1 Designing the Survey

3967 We followed the guidelines by Kitchenham and Pfleeger [195] on conducting per-
3968 sonal opinion surveys in software engineering to validate our survey. In developing
3969 our survey instrument, we shaped questions around each of our 5 dimensions and 34
3970 categories. To achieve this, we used Brooke’s SUS [61] as a loose inspiration and
3971 re-shaped the 34 categories around a question that imitates the style of wording of
3972 questions used in the SUS. Each dimension was marked a numeric question (Q#3–7),
3973 and alphabetic sub-questions were marked for each sub-dimension or category.

3974 We used closed questioning where respondents could choose an answer on a
3975 5-point Likert-scale (1=strongly disagree, 2=somewhat disagree, 3=neither agree
3976 nor disagree, 4=slightly agree and 5=strongly agree). Like Brooke’s study, each
3977 question alternated in positive and negative sentiment. Half of our questions were
3978 written where a likely common response would be in strong agreement and vice-
3979 versa for the other half, such that participants would have to “read each statement
3980 and make an effort to think whether they would agree or disagree with it” [61]. For
3981 example, the question regarding [B7] on API limitations was framed as: “*I believe it*
3982 *is important to know about what the limitations are on what the API can and cannot*
3983 *provide*” (Q4g), whereas the question regarding [C1] on domain concepts of the API
3984 was framed as: “*I wouldn’t read through theory about the API’s domain that relates*
3985 *theoretical concepts to API components and how both work together*” (Q5a).

3986 In addition, the remaining eight questions asked demographical information. An
3987 extra open question asked for further comments. The full survey is provided in Ap-
3988 pendix C.5 and anonymised survey data is available at <https://bit.ly/33siql1>.

3989 8.5.1.2 Evaluating the Survey

3990 After the first pass at designing questions was completed, we evaluated our survey
3991 on three researchers within our research group for general feedback. This resulted
3992 in minor changes, such as slight re-wording of questions and providing specific
3993 questions with examples (some with images). For example, the question regarding
3994 [A9] on an exhaustive list of all major components in the API was framed as “*I believe*

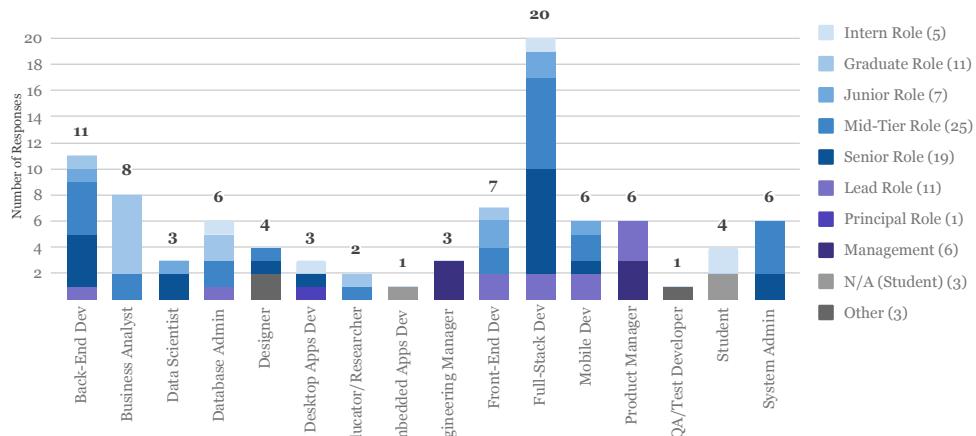


Figure 8.5: A wide variety of roles and seniority were observed in our respondents.

3995 an exhaustive list of all major components in the API without excessive detail would
 3996 be useful when learning an API” (Q3i) with the example “e.g., a computer vision
 3997 web API might list object detection, object localisation, facial recognition, and facial
 3998 comparison as its 4 components”.

3999 After this, we conducted reliability analysis using a test-retest approach on three
 4000 developers within our group seven weeks apart. Using the R statistical computation
 4001 environment [292], we conducted our analysis using the `IRR` package [128] (as
 4002 suggested in [149]) and resulted in an average intra-class correlation (ICC) of 0.63
 4003 which indicates a good overall index of agreement [78].

4004 8.5.1.3 Recruiting Participants

4005 Our target population for the study was application software developers with varying
 4006 degrees of experience (including those who and who have not used CVSs or related
 4007 tools before) and varying understanding of fundamental machine learning concepts.
 4008 We began by recruiting software developers within our research group using a group-
 4009 wide message sent on our internal messaging system. Of the 44 developers in our
 4010 group’s engineering cohort,⁵ 22 responses were returned, indicating an internal
 4011 response rate of 50.00%. Based on the 22 results from this internal trial, we
 4012 calculated the median time to our complete survey was just over 20 minutes.

4013 For external participant recruiting, we shared the survey on social media plat-
 4014 forms and online-discussion forums relevant to software development. We adopted
 4015 a non-probabilistic snowballing sampling where the participants, at the end of the
 4016 survey, were encouraged to share the survey link to others using *AddThis*.⁶ Ad-
 4017 ditionally, snowballing sampling was encouraged within members of our research
 4018 group who were asked to share the survey. This sampling approach resulted in 38
 4019 external responses. A further 44 participants were recruited via Amazon Mechanical

⁵Our research group’s engineering cohort consists of fully-qualified software engineers, with on average 5+ years industry experience.

⁶<https://www.addthis.com/> last accessed 7 January 2020.

4020 Turk⁷—often referred to as MTurk—which has been a successful approach adopted
4021 in previous software engineering surveys (e.g., [179]). To ensure our target demo-
4022 graphic was selected, we applied the participant filter option ‘Employment Industry
4023 - Software & IT Services’. An additional 13 responses were partially filled (on
4024 average at a completion rate of 43.23%). These partially completed responses were
4025 included in our analysis since they did yield some insight (see Section 8.7.2). As
4026 participants recruited via MTurk have a financial incentive to complete surveys,⁸ we
4027 ensured strict quality control was applied to each survey response we received. For
4028 example, 37 participants opened the survey but did not answer any questions; for
4029 this reason, all survey responses by these participants were discarded. We identified
4030 that 12 MTurk responses were filled out too quickly (where the median response
4031 time was under five minutes; well below the internal average of 20 minutes), and
4032 further analysis of these 12 responses indicated poor reading of the question, and
4033 thus poor responses; this was identified via our use of alternating positively- and
4034 negatively-worded questions. Thus, 12 MTurk responses were removed from the
4035 final analysis. Therefore, our final response rate yielded 104 responses of the total
4036 153 participants reached; an overall response rate of 67.97%.

4037 *8.5.1.4 Analysing Response Data*

4038 To analyse our response data, we produced a single score for each question’s 5-point
4039 response. In line with with Brooke’s SUS methodology [61], we subtracted one
4040 from the raw value of positive items, and subtracted the raw value from five for the
4041 negative items. This resulted in values on an ordinal scale of 0–4. We then averaged
4042 each response for every question and divide by four (i.e, now a 4-point scale) to
4043 obtain scores for each category. For example, two responses of *strongly agree*=5
4044 and one of *neither agree nor disagree*=3 were given to [A1] (positively worded);
4045 these values are mapped to 4 and 2, respectively, and are averaged (to 3.33) which is
4046 then divided by a maximum possible score of four, giving 0.84. We then discretise
4047 these calculated values into five intervals (as per Table 8.3, see Section 8.6.1.1) to
4048 interpret the findings; this is presented in Appendix C.1 under the ‘in-practice score’
4049 (IPS) for each category.

4050 Demographics for our survey were consistent in terms of the experience levels
4051 of developers who responded. 78% of respondents indicated they were professional
4052 programmers. Years of programming experience were: <1 year (3.30%); 1–5 years
4053 (41.76%); 6–10 years (35.16%); 11–15 years (9.89%); 16–20 years (5.49%); 21–
4054 30 years (3.30%); 31–40 years (1.10%); 41+ years (0.00%). A wide range of
4055 roles and seniority were listed by developers as presented in Figure 8.5, thereby
4056 indicating that our results include the different expectations of API documentation
4057 from a variety of sources. The highest role was a full-stack developer at either
4058 a mid-tier or senior role, followed by mid-tier or senior back-end developers and
4059 graduate and junior business analysts. Various managerial roles were also listed.

⁷<https://www.mturk.com/> last accessed 9 July 2020.

⁸A total budget of AUD\$600 was allocated for recruitment via MTurk, with each participant receiving between AUD\$3.50–\$10.00.

4060 Only five students (5.00%) responded in our study, two listing themselves as interns
4061 with one as an embedded applications developer. Most respondents were Australian
4062 (40.00%), Indian (26.70%) or from the United States (20.00%). Besides information
4063 technology services (30.77%), consulting and other software development (both at
4064 9.89%) were the most predominant industries listed by participants.

4065 **8.5.2 Empirical Application on Computer Vision Services**

4066 Once our taxonomy had been developed and assessed with developers, we performed
4067 an empirical application against three CVSSs: Google Cloud Vision [422], Amazon
4068 Rekognition [397] and Azure Computer Vision [436]. Our selection criteria in
4069 choosing these particular services to analyse is based on the prominence of the
4070 service providers in industry and the ubiquity of their cloud platforms (Google Cloud,
4071 Amazon Web Services, and Microsoft Azure) in addition to being the top three
4072 adopted vendors used for cloud-based enterprise applications [119]. In addition, we
4073 had conducted extensive investigation into the services' non-deterministic runtime
4074 behaviour and evolution profile in prior work [88] and have also identified developers'
4075 complaints about their incomplete documentation in a prior mining study on Stack
4076 Overflow [91].

4077 We began with an exploratory analysis of the presence of each dimension and
4078 its categories. Appendix C.2 displays all sources of documentation used; although
4079 we initially started on the respective services homepages [397, 422, 436], this search
4080 was expanded to other webpages hyperlinked. For each category, we listed the
4081 documentation's presence as either fully present, partially present or not present
4082 at all. This is shown in Appendix C.1 with the indication of (half-)filled circles or
4083 circle outlines for Google Cloud Vision (abbreviated to GCV), Amazon Rekognition
4084 (abbreviated to AWS), and Azure Computer Vision (abbreviated to ACV). Notes were
4085 taken for each webpage justifying the presence, and exact sources of documentation
4086 were listed when (partially) present. PDFs of each webpage were downloaded
4087 between 14–18 March 2019 for analysis. Analysis was performed manually by
4088 the lead author by manual inspection of the downloaded web pages (as PDFs) and
4089 presence of each item was noted by the lead author using an approach similar to
4090 Watson [371].

4091 **8.6 Taxonomy Analysis**

4092 In this section, we analyse investigating the taxonomy from two perspectives. Firstly,
4093 we contrast the ILS values, being an interpretation of the relevancy researchers have
4094 emphasised, against the IPS values found from the results of our survey (being
4095 an interpretation of what documentation artefacts developers value more). We are
4096 therefore able to identify the API documentation artefacts that are of high value
4097 to practitioners, but are yet to be deeply explored by researchers. Secondly, we
4098 contrast the IPS values against our assessment of CVSSs, and whether important API
4099 documentation artefacts have been included in popular services. We are therefore
4100 able to identify whether vendors have or have not already included these highly-

Table 8.3: Intervals of ILS (top) and IPS (bottom) values and frequencies.

Research Attention	Range	Frequency	Categories
Very Low	$0.00 \leq \text{ILS}([X_i]) < 0.14$	7	B4, B5, D6, B3, C1, D1, D2
Low	$0.14 \leq \text{ILS}([X_i]) < 0.29$	13	A1, A9, C3, D3, D4, E2, E3, E4, E5, B6, A7, A10, D5
Medium	$0.29 \leq \text{ILS}([X_i]) < 0.43$	9	B2, B7, A4, A12, E1, A3, A8, A11, C2
High	$0.43 \leq \text{ILS}([X_i]) < 0.57$	3	E6, B1, A2
Very High	$0.57 \leq \text{ILS}([X_i]) \leq 0.71$	2	A6, A5

Value to Developers	Range	Frequency	Categories
Very Low	$0.00 \leq \text{IPS}([X_i]) < 0.18$	0	-
Low	$0.18 \leq \text{IPS}([X_i]) < 0.36$	0	-
Medium	$0.36 \leq \text{IPS}([X_i]) < 0.53$	6	D4, B4, C3, C1, E4, B3
High	$0.53 \leq \text{IPS}([X_i]) < 0.71$	16	A4, B6, A2, D2, A6, E2, B5, D6, A8, B2, E6, A10, E5, D5, A9, D3
Very High	$0.71 \leq \text{IPS}([X_i]) \leq 0.89$	12	E3, A7, A3, C2, A12, B1, D1, A11, A1, E1, A5, B7

⁴¹⁰¹ valued documentation artefacts within their own APIs, and where existing areas of improvement lie.

⁴¹⁰³ 8.6.1 Exploring IPS and ILS Values

⁴¹⁰⁴ 8.6.1.1 IPS Results

⁴¹⁰⁵ IPS values indicate the extent to which developers agree with the statements made in ⁴¹⁰⁶ our survey, as calculated by the method described in Section 8.5.1.4. The interpretation ⁴¹⁰⁷ of these values are the documentation artefacts (categories) that developers *value* ⁴¹⁰⁸ the most. Thus collectively, these artefacts indicate the overall level of importance ⁴¹⁰⁹ towards specific API documentation requirements (dimensions).

⁴¹¹⁰ To interpret these values, we group the data from each of our survey’s 34 ⁴¹¹¹ statements (for each category) into an ordinal scale of five intervals. These intervals ⁴¹¹² indicate relative value to developers; a documentation artefact has *very low* value ⁴¹¹³ to developers, *low* value, *medium* value, *high* value, or *very high* value. Table 8.3 ⁴¹¹⁴ presents these intervals and frequencies of each, with the order of the categories ⁴¹¹⁵ shown in the last column indicating raw IPS values (least useful to most useful) ⁴¹¹⁶ before discretisation in ascending order.

⁴¹¹⁷ Practitioners tend to agree that each documentation artefact is important to have, ⁴¹¹⁸ and thus IPS values likely fall into the *High* or *Very High* intervals. Only six categories ⁴¹¹⁹ fall into the *Medium* interval and none fall into lower intervals. Developers ⁴¹²⁰ find technical support contact information [D4] to be of the lowest value (see Table ⁴¹²¹ 8.3), likely since developers tend to rely on crowd-sourced peer support through ⁴¹²² mediums such as Stack Overflow. They also see little value in: descriptions of the ⁴¹²³ types of end-users the API is intended for [B4]; documentation for non-technical ⁴¹²⁴ audiences [C3]; conceptual information relating the API back to its application do-

4125 main [C1]; structured navigation of the presented API documentation [E4]; and
4126 descriptions of the intended developers who should be using the API [B3].

4127 8.6.1.2 *ILS Results*

4128 ILS values indicate overall research attention of categories of our taxonomy through
4129 the proportion of papers in our SMS that investigated or reported various issues
4130 regarding a specific API documentation artefact. Collectively, each of these cat-
4131 egories combined form a dimension (labelled A–E) in a bottom-up approach (see
4132 Section 8.3.2.2). Each dimension (top-node) describes the requirements of good
4133 quality API documentation, while the category (leaf-node) is the specific API doc-
4134 umentation artefact that, collectively, form the requirement. A category with a
4135 high ILS value indicates that existing studies that there is substantial attention by
4136 researchers on this specific documentation artefact (or, collectively, requirement of
4137 good quality API documentation). Conversely, a lower ILS value indicates less
4138 attention reported on these categories (artefact) or dimensions (requirement) by the
4139 software engineering research community.

4140 To demonstrate the attention of these documentation artefacts within literature,
4141 we interpret the ILS values in a similar fashion to the IPS values. It is represented
4142 as a discretised value of intervals within a five-dimensional ordinal scale, where
4143 the attention on these artefacts in literature are one of: *very low* attention, *low*
4144 attention, *medium* attention, *high* attention, *very high* attention. Table 8.3 indicates
4145 the boundaries for each interval (as calculated by the highest ILS value of 0.71 divided
4146 by the five intervals) in addition to the frequency of categories appearing in each
4147 interval. The order of the categories shown in the last column indicate the ascending
4148 order (least research attention to most) of raw ILS values before discretisation. As
4149 shown, most of the artefacts (20) found in the taxonomy are discussed in literature
4150 disproportionately more than others (i.e., those that fall into the ‘low’ (13) or ‘very
4151 low’ (7) intervals), though the underlying reasons behind this should be considered
4152 on a case-by-case basis (see Section 8.7.3).

4153 There are only five categories that fall into the ‘high’ or ‘very high’ intervals,
4154 three of which fall under dimension [A], Descriptions of API Usage. Research atten-
4155 tion on a particular documentation artefact that is considered *Very High* gravitates
4156 towards code snippets [A5] and tutorials [A6]. Code snippets are the readiest form
4157 of API documentation for developers, representing exemplary nuggets of informa-
4158 tion for developers to rapidly digest singular components of the API’s functionality.
4159 While code snippets generally only reflect small portions of API functionality (gen-
4160 erally limited to 15–30 LoC), this is complimented by step-by-step tutorials. These
4161 may tie in multiple (disparate) components of API functionality to demonstrate de-
4162 velopment of more non-trivial applications. Therefore, unsurprisingly, research has
4163 substantially explored how best API developers can extract code snippets or write
4164 tutorials for these purposes in mind. This is followed by low-level reference doc-
4165 umentation [A2]—under the ‘high’ interval—whereby developers should document
4166 all client-facing implementation or usage aspects of their API (e.g., class, method,
4167 parameter descriptions etc.). Lastly, the entry-level purpose/overview of an API

4168 [B1] and consistency in the look and feel of the documentation throughout all of the
4169 API's official documentation [E6] are fall under the 'high' interval. API vendors
4170 must give motivation as to why a developer should choose a particular API over
4171 another, articulating the *need* of their API, presenting this and other documentation
4172 aspects in the easiest way for developers to consume.

4173 8.6.1.3 *Research Opportunities for High-Value Artefacts*

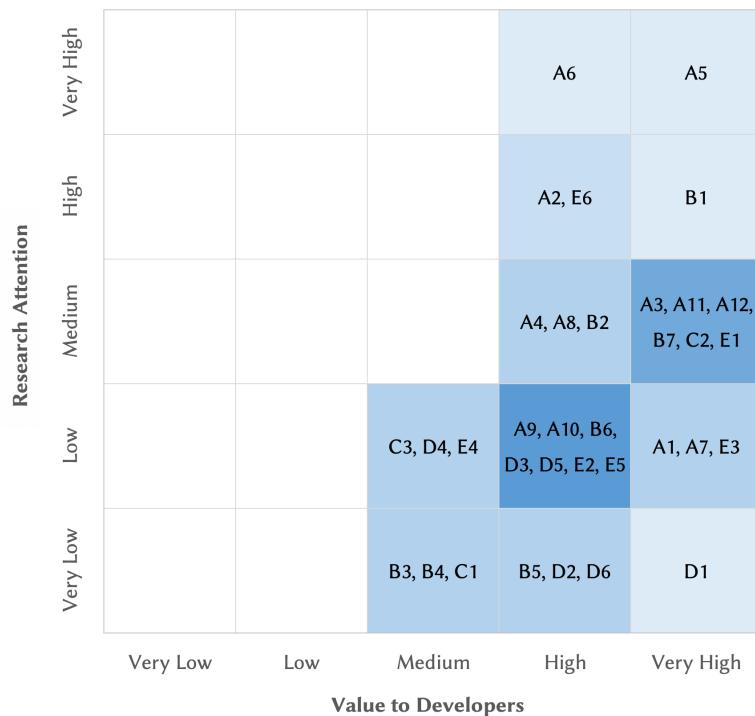


Figure 8.6: Value of API documentation artefacts to developers (IPS) vs their research attention (ILS). Colour intensity represents greater number of categories in each intersection

4174 In this section, we explore the ILS and IPS values as two distinct indicators of
4175 research exploration that would provide the most value to practitioners. We then
4176 provide a qualitative discussion by inspecting the intersection of categories at each
4177 respective interval identified by our SMS and survey study. Thus, we are able to
4178 determine documentation artefacts (categories) and requirements (dimensions) that
4179 provide the *greatest value* to developers but have not gained proportional attention
4180 in the software engineering literature when compared to other artefacts, and vice-
4181 versa. Graphically, we represent these intersections within a five-by-five matrix with
4182 intervals of the IPS (x axis) plotted against intervals of the ILS (y axis). Intersections
4183 between the two are listed for each category within the taxonomy. This is presented
4184 in Figure 8.6.

4185 There is a distinction between (very)-highly valued documentation artefacts
4186 whose research attention is (very)-low, as presented in the bottom-right of Figure 8.6.
4187 Most notably, we find that developers find Existence of Support Artefacts [D] a highly

4188 valued API documentation requirement, but there still exists a substantial gap in
 4189 existing literature into this requirement. For example, besides category [D4] (which
 4190 is of only *Medium* value to developers), less research has explored all other dimension
 4191 [D] categories (though there may be understandable reasons as to why, as detailed in
 4192 Section 8.7.3). Furthermore, developers highly value detailed Descriptions of API
 4193 Usage [A] through many documentation artefacts, notably quick-start guides [A1],
 4194 downloadable sample applications [A7], exhaustive list of major components [A9],
 4195 and system requirements to use the API [A10]. Such artefacts emphasise the need
 4196 for developers to rapidly pick-up a new API; however, the best ways to provide such
 4197 information is still open to further investigation in literature.

4198 Conversely, the top-right of Figure 8.6 emphasises (very)-highly researched
 4199 artefacts that are of (very)-high value to developers. Here we see that Descriptions
 4200 of API Usage [A] is the most-researched requirement, with code snippets [A5] being
 4201 an API usage artefact that is both most-researched and of highest value. Hence,
 4202 this demonstrates how many existing studies have an empirical basis on software
 4203 developers (e.g., via surveys or interviews; see Figure 8.3)—code snippets is a well-
 4204 researched artefact since most developers agree to its need in the documentation of
 4205 APIs. Therefore, it is clear to see how the correlation between the respective ILS
 4206 and IPS values for [A5] are high. However, if we look at other areas of our taxonomy,
 4207 such as [A12], [B7], [D3], [E3] or [E5], we find that developers do indeed desire these
 4208 aspects of API documentation, and, consequently, demand usage descriptions, design
 4209 rationale descriptions, support artefacts, or good presentation of the documentation
 4210 to be a necessary requirement of good quality API documentation. Thus, these
 4211 aspects have not gained proportional attention in literature, thereby highlighting
 4212 future research potential.

4213 8.6.2 Triangulating IPS, ILS and Computer Vision

4214 To interpret our comparison of IPS values with CVSSs, we introduce a calculated
 4215 ‘presence score’ for each category. As discussed in Section 8.5.2, we empirically
 4216 evaluate each category of our taxonomy with three CVSSs: Azure Computer Vision
 4217 (ACV), Amazon Rekognition (AWS) and Google Cloud Vision (GCV). We indicate
 4218 whether the respective API documentation artefact is present, partially present, or
 4219 nor present (as listed in Appendix C.1). To interpret this data, we assign a full circle
 4220 (\bullet) for present, half-circle (\circ) for partially present and an empty circle (\circ) for not
 4221 present. Combinations of presence for each category per service are indicated with
 4222 the three circles of varying shade. For example, [A1] has a presence score of $\bullet\bullet\circ$
 4223 because it was found to be present in both GCV and ACV but only partially present
 4224 in AWS; [B3] has a presence score of $\bullet\circ\circ$ because it was only found to be partially
 4225 present in GCV, etc. For a list of full presence values, see Appendix C.1.

4226 We illustrate which artefacts industry vendors provide developers with and the
 4227 artefact’s respective developer value using this combination of three circles. Using
 4228 a similar approach to the previous section, these results are presented in a ten-by-
 4229 five matrix as illustrated in Figure 8.7. If only one service fully implements a
 4230 documentation artefact of (very)-high value to developers ($\bullet\circ\circ$), if one or two

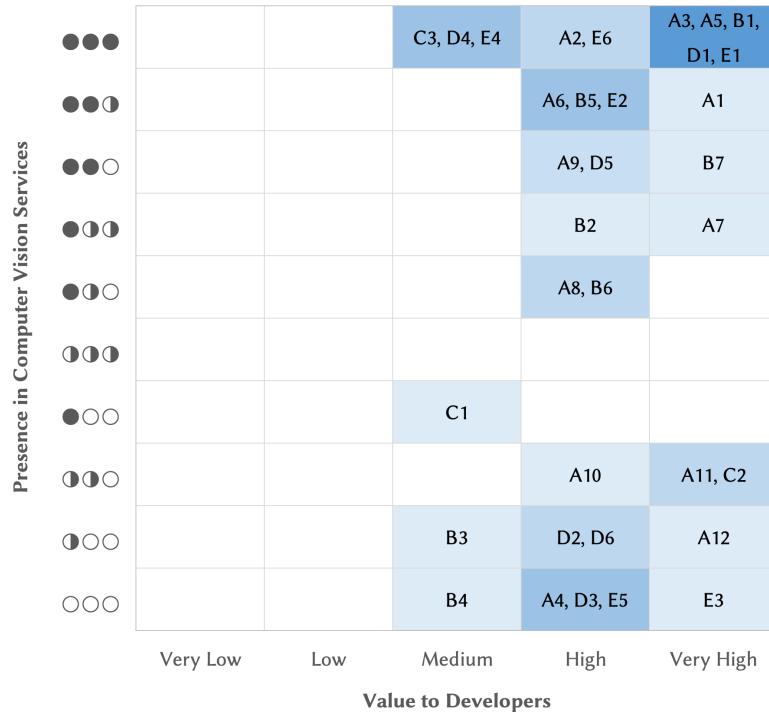


Figure 8.7: Value of API documentation artefacts to developers (IPS) vs their presence in CVSs. Colour intensity represents greater number of categories in each intersection.

⁴²³¹ services partially implement the artefact (**●○○** and **●●○**) or if none do (**○○○**),
⁴²³² then we believe there is room for improvement for service vendors to improve their
⁴²³³ documentation and include these artefacts.

⁴²³⁴ In this instance, we can see 10 categories listed in Figure 8.7 that developers
⁴²³⁵ feel are important but are not fully implemented across all three CVS vendors.
⁴²³⁶ This is especially the case for dimensions [A] (Descriptions of API Usage) and [D]
⁴²³⁷ (Existence of Support Artefacts), corroborating our findings with existing gaps in
⁴²³⁸ literature under Section 8.6.1.3. In other words, while both the goals of existing
⁴²³⁹ studies and CVS vendors have emphasised the need for artefacts such as code-
⁴²⁴⁰ snippets [A5], tutorials [A6], and entry-points to the API [B1], less attention is
⁴²⁴¹ given to by *both* literature and vendors on the same, (very-)highly valued aspects to
⁴²⁴² developers (e.g., troubleshooting hints [D2], licensing information [D6] or links to
⁴²⁴³ related components [E3]).

⁴²⁴⁴ Furthermore, from our analysis, we can see areas with which the research com-
⁴²⁴⁵ munity has and has *not* paid extensive attention to. We still see that vendors have
⁴²⁴⁶ paid attention to artefacts even where there has been less research attention, namely
⁴²⁴⁷ [D1] (FAQs), [B5] (success stories), [A7] (downloadable sample applications), [A1]
⁴²⁴⁸ (quick-start guides), [E2] (forums), [D5] (printable guides), and [A9] (API compo-
⁴²⁴⁹ nent lists). These seven categories are of (very) high value to developers but research
⁴²⁵⁰ attention on these topics are (very) low; however, their presence score within CVSs
⁴²⁵¹ are **●●○** or greater. Hence, we can see that vendors address developer's concerns

despite the lack of attention by software engineering researchers in these areas, and thus future research potential to better serve developers and ensure vendors' implementation of these documentation artefacts is evident.

From the above, we can therefore conclude that the vendors' documentation largely covers a majority of API documentation requirements. However, there still remains opportunity for improvement to API documentation by either vendors and/or the research community: that is, low research attention on documentation artefacts that present high value to developers which are *also* generally missing from vendor documentation. To explore this aspect, we triangulate the documentation artefacts (categories) that have a low or very low research attention and that are only present in one service, partially present in one or two, or not present at all. This results in three documentation requirements that warrant further exploration by industry vendors or the research community (see Table 8.4).

8.6.3 Recommendations Resulting from Analysis

In this section, we triangulate the taxonomy developed from literary sources, the developer survey on this taxonomy to understand its efficacy in-practice, and the application of the taxonomy to CVSs to provide several recommendations for both service providers and researchers. Our recommendations are based both on extrapolations of our findings, our prior work, and existing experience with such work.

8.6.3.1 Recommendations for vendors

Table 8.4 emphasises how service vendors still lack key documentation requirements of critical importance to developers that are still widely under-researched in software engineering literature. The largest of these requirements are the need for vendors to provide additional support artefacts [D] and the need for vendors to present this in a way that's most digestible for developers to understand [E]. A list of detailed suggestions for vendors are provided in Appendix C.4; here we discuss generalised findings on a sample of key artefacts.

For example, no services assessed had any form of diagrammatic overview of their APIs at a high-level [D3], thereby indicating how various components of their APIs work together, such as how specific endpoints work or an overview of the lifecycle of the technical domain behind these endpoints (i.e., label/train/infer/re-train), thereby incorporating conceptual relationships behind the API [C1]. For instance, an interactive overview of the developer's need to pre-process their data, send it to the service, and post-process the response data would help developers understand how the service better fits into the 'flow' of their application. Moreover, we failed to find lower-level diagrammatic overviews of the client SDKs—such as a UML diagram—that developers find very useful. We strongly advise vendors to provide diagrams illustrating the service within context to help support existing written documentation.

Troubleshooting hints [D2] are also a valuable support artefact, but were only found for AWS's video processing endpoints. As our prior work shows, developers are likely to question what aspects of the service can and cannot do, such as the types

Table 8.4: Documentation artefacts of high value to developers that have less attention in software engineering literature and are under-documented in CVSs. Documentation requirements (i.e., dimensions) separated by rules.

Artefact	Value	Research Attention	Presence in Computer Vision Services
[A10] Documenting API's minimum system requirements and/or dependencies	High	Low: 5 studies (23%)	Score=1.0: No dedicated web pages found for this artefact in any service. Dependencies for client libraries embedded within GCV and ACV quick-start guides [425, 439]. Other system requirements not listed.
[D2] Troubleshooting hints	High	Very Low: 2 studies (10%)	Score=0.5: Only found in AWS's video recognition service [407], but no troubleshooting tips found for non-video image recognition.
[D3] Diagrammatic representation of API [D6] Licensing Information	Very High Very High	Low: 3 studies (14%) Very Low: 1 study (5%)	Score=0.0: Not found for any service. Score=0.5: Partially present only in ACV [443]; information is non-specific to the licensing terms of ACV exclusively.
[E3] Quick-links to other relevant components [E5] Visualised map of navigational paths	Very High Very High	Low: 3 studies (14%) Low: 3 studies (14%)	Score=0: Not found for any service. Score=0: Not found for any service.

of labels it can find, or how to make it focus on specific ontologies when an input image is provided; e.g., time of day (day vs night) location (indoors vs outdoors) or the subject of the image (dog vs cat) [91]. Troubleshooting in identifying service evolution [91] would also be important, since developers are likely to overlook subtle (but application-breaking) changes to response data, such as labels introduced/removed or confidence changes. Therefore, vendors must document detailed troubleshooting suggestions on their websites on how best to resolve discrepancies in the results found from these services. This could easily be tied in with [A12] to incorporate usage description requirements when errors are presented to users and how to deal with them; also largely missing from existing documentation.

Another important aspect is the need to make documentation of one component more easily relatable to other parts of the documentation [E3]. Again, no service provided quick-links to related documentation; an example here could be links to definitions of domain-specific terminology [C2] to help developers with the learning process of adopting these new generation of APIs (e.g., the ‘score’ field could be linked back to a video explaining the concept of probability within the services’ guesses).

8.6.3.2 Recommendations for researchers

As shown in Table 8.4, we see that there are cases of (very) high-value documentation artefacts (to practitioners) in which literature has not paid great attention to. For example, for the requirement of API usage description [A], practitioners agree that both code snippets [A5] and documenting system requirements to use the API [A10] are of, at least, high value. However, while code snippets has had *consistent* attention within the software engineering research community (i.e., 15 papers spanning 1998–2019), we see that system requirements documentation only gained fluctuating interest by researchers (i.e., predominantly in the 2000s, with two further papers in the last three years). Thus, five papers investigating *some* aspects on this artefact may not cover *all* its aspects; for example, we may have identified a *need* to document these requirements and dependencies, but does this mean we know *all* aspects on how to produce them, the best way to *communicate* them, and the most efficient means for developers to *consume* that information? Contrasting this artefact against the 15 papers on code snippets, we see two documentation artefacts of at least high value to practitioners, yet, evidently, researchers have paid attention to one over the other.

As Figure 8.6 shows, the need for additional support [D] within documentation is the largest requirement that *may* be an indicator for further research in this domain (see Section 8.7.3). Notably, RQ2 of our SMS identified the methodologies and data collection techniques by which our existing understanding of API documentation requirements were gathered; as demonstrated through Figure 8.3, a majority of our understanding is grounded through the opinions of developers, namely evaluation research using direct techniques. Too many studies are shown to rely on a handful of data collection techniques (interviews and questionnaires, shadowing and observation, think-aloud sessions) and a stronger emphasis for indirect and independent

4337 techniques is needed moving forward; there is therefore a gap in literature on *other*
4338 types of data collection techniques that may provide different insights into satisfying
4339 the documentation requirements within our taxonomy.

4340 For example, we see [A9] (exhaustive list of major API components) as a high-
4341 value documentation artefact that satisfies the requirement of the API usage descrip-
4342 tion [A]. However research attention is lower. A validation research paper could
4343 propose a method to generate a baseline list of these components through an inde-
4344 pendent technique, such mining the API codebase for its major components through
4345 class usage (static analysis) or analysing an existing work database or tool use logs
4346 to see which components developers have accessed the most. This would satisfy
4347 the need for the documentation artefact, bolstering the API usage requirement and
4348 exploring new techniques to do so.

4349 Few philosophical papers result in a lack of insight into completely new ways of
4350 exploring API documentation. Further exploration into this type of research may help
4351 us devise a whole new framework of producing API documentation. For example,
4352 as shown by developers and vendors, quick-start guides [A1] are highly valued,
4353 and well-documented in CVSs. But literature does not provide any vocabulary
4354 or frameworks into how best to develop such guides. Involving both software
4355 engineering researchers and developers through a brainstorming or focus group to
4356 conceptualise, devise, and refine such a framework may be a worthwhile study to
4357 better improve our understanding of quick-start guides whilst also exploring new
4358 approaches to research new guidelines.

4359 Beyond requirement [A], another insight identified is the need for developers to
4360 have visualised maps of navigational paths [E5] which is not yet provided by any of
4361 the CVS providers investigated. With the low ILS value in this category (14% or
4362 3 studies), we see a potential research topic for future exploration. For example, if
4363 research can demonstrate that such visualised maps are not just something developers
4364 desire, but can make them *more effective* in their day-to-day work, then this could be
4365 a strong case made to vendors to improve the presentation of their documentation.

4366 Thus, as we have shown in these sample recommendations, many potential
4367 studies and research directions can stem by exploring the discrepancies of API
4368 documentation in literature, in practice, and their presence in CVSs (i.e., as a sample
4369 case study) when assessed on a case-by-case basis. The method researchers decide
4370 upon depends the research questions they wish to address; thus, observations we
4371 present in Figure 8.3 may trigger fruitful reasoning about approaches future research
4372 could take, however inferring methodological gaps will need to be compatible with
4373 research goals. Thus, mapping these discrepancies to gaps in the techniques used in
4374 studies to devise of novel ways to improve API documentation whilst also exploring
4375 new methodologies should be balanced carefully by researchers.

4376 8.7 Threats to Validity

4377 8.7.1 Internal Validity

4378 Threats to *internal validity* represent internal factors of our study which affect
4379 concluded results. Kitchenham and Charters' guidelines on producing systematic
4380 reviews [194] suggest that researchers conducting reviews should discuss the review
4381 protocol, inclusion decisions, data extraction with a third party. Within this study,
4382 we discussed our protocols with other researchers within our research group and
4383 utilised test-retest reliability. Further assessments into reliability would involve an
4384 assessment of the review and extraction processes, which can be investigated using
4385 inter-rater reliability measures. Guidelines suggested by Garousi and Felderer [130]
4386 describe methods for independent analysis and conflict resolution could help resolve
4387 this.

4388 As stated in Section 8.3.2, we utilised a systematic software engineering tax-
4389 onomy development method by Usman et al. [360]. Two additional taxonomy
4390 validation approaches proposed by Usman et al. were not considered in our work:
4391 benchmarking and orthogonality demonstration. To our knowledge, there are no
4392 other studies that classify existing API documentation studies into a structured tax-
4393 onomy, and therefore we are unable to benchmark our taxonomy against others. We
4394 would encourage the research community to conduct a replication of our work and
4395 investigate whether our taxonomy classification approaches are replicable to ensure
4396 that categories are reliable and the dimensions fit the objectives of the taxonomy.
4397 Moreover, we did not investigate orthogonality demonstration as our primary goals
4398 for this work were to investigate the efficacy of the taxonomy by practitioners and
4399 in-practice, with reference to our wider research area of intelligent CVSSs. Therefore,
4400 we solely adopted the utility demonstration approach in two detailed experiments
4401 (Sections 8.5 and 8.6) to analyse the efficacy of our taxonomy and identify potential
4402 improvements for these services' API documentation.

4403 8.7.2 External Validity

4404 Threats to *external validity* concern the generalisation of our observations. Our
4405 SMS has used a broad range of sources however not all papers contributing to API
4406 documentation may have been found or captured within the taxonomy. While we
4407 attempted to include as many papers as we could find in our study, some papers may
4408 have been filtered out due to our exclusion criteria. For example, there are studies
4409 we found that were excluded as they were not written in English, and these excluding
4410 factors may alter our conclusions, introducing conflicting recommendations. How-
4411 ever, given the consistency of these trends within the studies that were sourced, we
4412 consider this a low likelihood.

4413 Online documentation of APIs are non-static, and may evolve using contribu-
4414 tions from both official sources and the developer community (e.g., via GitHub).
4415 We downloaded the three service's API documentation in March of 2019—it is
4416 highly likely that new documentation may have been added since or modified since
4417 publication. A recommendation to mitigate this would be to re-evaluate this study

4418 once intelligent CVSs have matured and become even more mainstream in developer
4419 communities.

4420 Unless significant inducements are offered, Singer et al. [331] report that a
4421 consistent response rate of 5% has been found in software engineering questionnaires
4422 distributed and in information systems the median response rates for surveys are 60%
4423 [29]. We observe that low response rates may adversely effect the findings of our
4424 survey, typically as software engineers find little time to do them [331]. When
4425 compared to typical software engineering studies, our response rate of 67.97% was
4426 likely successful due to designing and carefully testing succinct, unambiguous and
4427 well-worded questions with researchers within our research group. All adjustments
4428 made from the pilot study due to unexpected poor quality of the questionnaire have
4429 been reported and explained in Section 8.5.1.2. However, further improvements
4430 could be made to increase this response rate.

4431 The survey reached 82 external and 22 internal participants. This yielded a total
4432 of 104 participants. However, only 91 participants fully completed the survey and,
4433 on average, those who only partially completed the survey completed 43.23% of
4434 all questions. Therefore, demographic data for these participants is largely missing.
4435 To verify the reliability of partially submitted responses, we calculated the average
4436 response of each item in our survey (i.e., question) for all fully completed results and
4437 all partially completed results. All partially completed questions, except [B7], were
4438 within 1 standard deviation from the mean, and therefore we believe the 13 partial
4439 results to be valid when excluding B7. Even if these partial results are excluded, our
4440 full-response participant count of 91 is still comparable to existing studies, such as
4441 Nykaza et al. [264] (57 participants), Robillard and Deline [305] (80 participants),
4442 or [304] (83 participants). Therefore, given these comparable numbers, we believe
4443 this does not compromise validity of our results.

4444 We also adopt research conducted in the field of questionnaire design, such as
4445 ensuring all scales are worded with labels [205] and have used a summatting rating
4446 scale [336] to address a specific topic of interest if people are to make mistakes in
4447 their response or answer in different ways at different times. This approach was
4448 also extended using alternating positive and negative sentiment for each question—
4449 as multiple studies have shown [62, 315], this approach helps reduce poor-quality
4450 responses by minimising extreme responses and acquiescence biases.

4451 **8.7.3 Construct Validity**

4452 Threats to *construct validity* relates to the degree by which the data extrapolated
4453 in this study sufficiently measures its intended goals. Our interpretation of the
4454 ILS (as given in Sections 8.4 and 8.6.1.2) is reported as the proportion of papers
4455 whose research investigates or explores issues regarding the aspects of specific API
4456 documentation artefacts (i.e., categories in the taxonomy) that, collectively, comprise
4457 the requirements of good API documentation (i.e., dimensions in the taxonomy).
4458 Every effort has been made in this work to provide a constructive analysis on the API
4459 documentation landscape, however, the studies that comprise the ILS may differ in
4460 their intent toward a specific documentation artefact. For example, some studies may

4461 have distinct goals to extensively study *how* code snippets [A5] specifically improve
4462 developer productivity (e.g., through interviews or by observational studies), while
4463 others may just reflect that code snippets are a commonly-used artefact self-reported
4464 by developers (e.g., through a survey). Thus, the interpretation of the ILS may range
4465 between deep exploration of an artefact or whether a study mentions the artefact
4466 without any attempts to thoroughly investigate it. For this reason, we suggest that a
4467 high ILS value for a category within the taxonomy suggests that the documentation
4468 artefact is within the attention of the research community, and that subsequent
4469 attention **may** be required for those artefacts with low ILS values as a *potential*
4470 *indicator* for future research (i.e., it also may *not*). However, each artefact with a
4471 low ILS (but high IPS) would need to be carefully examined in isolation to evaluate
4472 whether future research is indeed warranted, and how that research can be conducted
4473 with the ultimate goal to assist practitioners.

4474 Automatic searching was conducted in the SMS by choice of three popular
4475 databases (see Section 8.3.1). As a consequence of selecting multiple databases,
4476 duplicates were returned. This was mitigated by manually curating out all duplicate
4477 results from the set of studies returned. Additionally, we acknowledge that the lack
4478 manual searching of papers within particular venues may be an additional threat due
4479 to the misalignment of search query keywords to intended papers of inclusion. Thus,
4480 our conclusions are only applicable to the information we were able to extract and
4481 summarise, given the primary sources selected.

4482 While we have investigated the application of this taxonomy using a user study
4483 (Section 8.5.1), we would like to explore a controlled study of developers to assess
4484 how improved and non-improved API documentation impacts developer productiv-
4485 ity. The outcome of this work can help design a follow-up experiment, consisting of a
4486 comparative controlled study [321] that capture firsthand behaviours and interactions
4487 toward how software engineers approach using a CVS with and without our taxon-
4488 omy applied. This can be achieved by providing ‘mock’ improved documentation
4489 with the suggested improvements included in this work. Such an experiment could
4490 recruit a sample of developers of varying experience (from beginner programmer
4491 to principal engineer) to complete a certain number of tasks under a comparative
4492 controlled study, half of which will (a) develop using the improved ‘mock’ docu-
4493 mentation, and the other half will (b) develop with the *as-is/existing* documentation.
4494 From this, we can compare if the taxonomy makes improvements by capturing met-
4495 rics and recording the sessions for qualitative analysis. Visual modelling can be
4496 adopted to analyse the qualitative data using matrices [98], maps and networks [318]
4497 as these help illustrate any causal, temporal or contextual relationships that may exist
4498 to map out the developer’s mindset and difference in approaching the two sets of
4499 designs of the same tasks.

4500 8.8 Conclusions & Future Work

4501 The emergence of AI-based intelligent components present significant challenges to
4502 our existing understanding of traditional API documentation. The inherent prob-
4503 abilistic and non-deterministic nature of these components means that developers

4504 must shift their mindset of conventional APIs, and vendors of these services must
4505 similarly shift the mindset of documenting their APIs using traditional means. With-
4506 out adapting to the new mental model (of the vendors designing these services) and
4507 by vendors presenting poor or incomplete (traditional) documentation that is not
4508 compatible with these next-generation components, developers face many struggles.
4509 They fail to grasp how to properly understand how these services work, seeking fur-
4510 ther documentation or support from their peers on forums on such as Stack Overflow
4511 [91]. This ultimately hinders developers' productivity and thus adversely affects the
4512 internal quality of the applications that they build.

4513 This study has explored the artefacts and means by which traditional API doc-
4514 umentation is studied through the use of an SMS of 4,501 studies, identifying 21
4515 key works. From this, we synthesised a taxonomy of the various documentation
4516 artefacts that improves API documentation quality, and thus collectively synthesis-
4517 ing the requirements of good API documentation. Furthermore, we also capture the
4518 most commonly used analysis techniques used in the academic literature to under-
4519 stand the means by which the goals of these studies resulted in their findings. We
4520 then validate our taxonomy against developers to assess its efficacy with practitio-
4521 ners, and conduct a heuristic evaluation against three popular CVSs. We determine
4522 that developers demand certain documentation artefacts more than others, since not
4523 all documentation artefacts are equally valued. We map the value (to developers)
4524 of these artefacts against their exposure within the software engineering literature,
4525 thereby highlighting the gaps by which future research could expand upon. Fur-
4526 thermore, we present a similar mapping against how well the coverage CVSs have
4527 incorporated such artefacts into their own API documentation, thus highlighting
4528 that while industry vendors cover most documentation artefacts that may not be in
4529 the interest to researchers, some artefacts with low research interest are still largely
4530 missing (see Table 8.4). We therefore provide several generalised recommendations
4531 to vendors and the wider research community to explore how best these artefacts
4532 can be better addressed and incorporated into further research, thus improving our
4533 understanding of the requirements of good API documentation.

4534 Future extensions of our work may involve a restricted systematic literature
4535 review in API documentation artefacts, and many suggestions are further detailed
4536 in Section 8.7. Further, a review into the techniques of these primary studies may
4537 extend the mapping we conducted in this work, by evaluating the the effectiveness of
4538 the various approaches used in each study and assessing these against the proposed
4539 conclusions of each study.

4540 The findings of our work provides a solid baseline for improving the documen-
4541 tation of non-deterministic software, such as CVSs. While our aim is to eventually
4542 improve the quality of API documentation, the ultimate goal is to improve the
4543 software engineer's experience of non-deterministic and abstracted AI-based com-
4544 ponents, such as intelligent web services (IWSs). We hope the guidelines from this
4545 extensive study help both software developers and API providers alike by using our
4546 taxonomy as a go-to checklist for what should be considered in documenting any
4547 API.

CHAPTER 9

4548

4549

4550 Using a Facade Pattern to combine Computer Vision Services[†]

4551

4552 **Abstract** Intelligent computer vision services, such as Google Cloud Vision or Amazon
4553 Rekognition, are becoming evermore pervasive and easily accessible to developers to build
4554 applications. Because of the stochastic nature that ML entails and disparate datasets used in
4555 their training, the outputs from different computer vision services varies with time, resulting
4556 in low reliability—for some cases—when compared against each other. Merging multiple
4557 unreliable API responses from multiple vendors may increase the reliability of the overall
4558 response, and thus the reliability of the intelligent end-product. We introduce a novel
4559 methodology—inspired by the proportional representation used in electoral systems—to
4560 merge outputs of different intelligent computer vision API provided by multiple vendors.
4561 Experiments show that our method outperforms both naive merge methods and traditional
4562 proportional representation methods by 0.015 F-measure.

4563 9.1 Introduction

4564 With the introduction of intelligent web services (IWSs) that make machine learning
4565 (ML) more accessible to developers [300, 367], we have seen a large growth of
4566 intelligent applications dependent on such services [65, 135]. For example, consider
4567 the advances made in computer vision, where objects are localised within an image
4568 and labelled with associated categories. Cloud-based computer vision services
4569 (CVSs)—e.g., [397, 410, 418, 422, 431, 432, 436, 485]—are a subset of IWSs.
4570 They utilise ML techniques to achieve image recognition via a remote black-box
4571 approach, thereby reducing the overhead for application developers to understand
4572 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.

4573 and training of the machine-learnt algorithms is offloaded to the cloud, developers
4574 simply send RESTful API requests to do the recognition. There are, however, inherit
4575 differences and drawbacks between traditional web services and IWSs, which we
4576 describe with the motivating scenario below.

4577 **9.1.1 Motivating Scenario: Intelligent vs Traditional Web Services**

4578 An application developer, Tom, wishes to develop a social media Android and iOS
4579 app that catalogues photos of him and his friends, common objects in the photo,
4580 and generates brief descriptions in the photo (e.g., all photos with his husky dog,
4581 all photos on a sunny day etc.). Tom comes from a typical software engineering
4582 background with little knowledge of computer vision and its underlying concepts.
4583 He knows that intelligent computer vision web APIs are far more accessible than
4584 building a computer vision engine from scratch, and opts for building his app using
4585 these cloud services instead.

4586 Based on his experiences using similar cloud services, Tom would expect consistency
4587 of the results from the same API and different APIs that provide the same (or
4588 similar) functionality. As an analogy, when Tom writes the Java substring method
4589 "doggy".substring(0, 2), he expects it to be the same result as the Swift equivalent
4590 "doggy".prefix(3). Each and every time he interacts with the substring
4591 method using either API, he gets "dog" as the response. This is because Tom is
4592 used to deterministic, rule-driven APIs that drive the implementation behind the
4593 substring method.

4594 Tom's deterministic mindset results in three key differentials between a traditional
4595 web services and an IWS:

4596 **(1) Given similar input, results differ between similar IWSs.** When Tom
4597 interacts with the API of an IWS, he is not aware that each API provider trains
4598 their own, unique ML model, both with disparate methods and datasets. These
4599 IWSs are, therefore, nondeterministic and data-driven; input images—even
4600 if they contain the same conceptual objects—often output different results.
4601 Contrast this to the substring method of traditional APIs; regardless of what
4602 programming language or string library is used, the same response is expected
4603 by developers.

4604 **(2) Intelligent responses are not certain.** When Tom interprets the response
4605 object of an IWS, he finds that there is a ‘confidence’ value or ‘score’. This
4606 is because the ML models that power IWSs are inherently probabilistic and
4607 stochastic; any insight they produce is purely statistical and associational [280].
4608 Unlike the substring example, where the rule-driven implementation provides
4609 certainty to the results, this is not guaranteed for IWSs. For example, a picture
4610 of a husky breed of dog is misclassified as a wolf. This could be due to
4611 adversarial examples [345] that ‘trick’ the model into misclassifying images
4612 when they are fully decipherable to humans. It is well-studied that such
4613 adversarial examples exist in the real world unintentionally [113, 206, 283].

4614 **(3) Intelligent APIs evolve over time.** Tom may find that responses to processing
4615 an image may change over time; the labels he processes in testing may evolve

4616 and therefore differ to when in production. In traditional web services, evo-
4617 lution in responses is slower, generally well-communicated, and usually rare
4618 (Tom would always expect "dog" to be returned in the substring example).
4619 This has many implications on software systems that depend on these APIs,
4620 such as confidence in the output and portability of the solution. Currently, if
4621 Tom switches from one API provider to another, or if he doesn't regularly test
4622 his app in production, he may begin to see a very different set of labels and
4623 confidence levels.

4624 9.1.2 Research Motivation

4625 These drawbacks bring difficulties to the intended API users like Tom. We identify a
4626 gap in the software engineering literature regarding such drawbacks, including: lack
4627 of best practices in using IWSs; assessing and improving the reliability of APIs for
4628 their use in end-products; evaluating which API is suitable for different developer
4629 and application needs; and how to mitigate risk associated with these APIs. We
4630 focus on improving reliability of CVSs for use in end-products. The key research
4631 questions in this paper are:

- 4632 **RQ1:** Is it possible to improve reliability by merging multiple CVS results?
4633 **RQ2:** Are there better algorithms for merging these results than currently in
4634 use?

4635 Previous attempts at overcoming low reliability include triple-modular redundancy
4636 [225]. This method uses three modules and decides output using majority
4637 rule. However, in CVSs, it is difficult to apply majority rule: these APIs respond with
4638 a list of labels and corresponding scores. Moreover, disparate APIs ordinarily output
4639 different results. These differences make it hard to apply majority rule because the
4640 type of outputs are complex and disparate APIs output different results for the same
4641 input. Merging search results is another technique to improve reliability [330]. It
4642 normalises scores of different databases using a centralised sample database. Nor-
4643 malising scores makes it possible to merge search results into a single ranked list.
4644 However, search responses are disjoint, whereas they are not in the context of most
4645 CVSs.

4646 In this paper, we introduce a novel method to merge responses of CVSs, using
4647 image recognition APIs endpoints as our motivating example. Section 9.2 describes
4648 naive merging methods and requirements. Section 9.3 gives insights into the struc-
4649 ture of labels. Section 9.4 introduces our method of merging computer vision labels.
4650 Section 9.5 compares precision and recall for each method. Section 9.6 presents
4651 conclusions and future work.

4652 9.2 Merging API Responses

4653 Image recognition APIs have similar interfaces: they receive a single input (image)
4654 and respond with a list of labels and associated confidence scores. Similarly, other
4655 supervised-AI-based APIs do the same (e.g., detecting emotions from text and

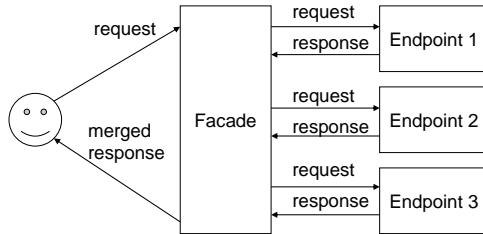


Figure 9.1: The user sends a request to the facade; this request is propagated to the relevant APIs. Responses are merged by the facade and returned back to the user.

4656 natural language processing [433, 486]). It is difficult to apply majority rule on such
4657 disparate, complex outputs. While the outputs by *multiple* AI-based API endpoints
4658 is different and complex, the general format of the output is the same: a list of labels
4659 and associated scores.

4660 9.2.1 API Facade Pattern

4661 To merge responses from multiple APIs, we introduce the notion of an API facade.
4662 It is similar to a metasearch engine, but differs in their external endpoints. The
4663 facade accepts the input from one API endpoint (the facade endpoint), propagates
4664 that input to all user-registered concrete (external) API endpoints simultaneously,
4665 then ‘merges’ outputs from these concrete endpoints before sending this merged
4666 response to the API user. We demonstrate this process in Figure 9.1.

4667 Although the model introduces more time and cost overhead, both can be miti-
4668 gated by caching results. On the other hand, the facade pattern provides the following
4669 benefits:

- 4670 • **Easy to modify:** It requires only small modifications to applications, e.g.,
4671 changing each concrete endpoint URL.
- 4672 • **Easy to customise:** It merges results from disparate and concrete APIs ac-
4673 cording to the user’s preference.
- 4674 • **Improves reliability:** It enhances reliability of the overall returned result by
4675 merging results from different endpoints.

4676 9.2.2 Merge Operations

4677 The API facade is applicable to many use cases. However, this paper focuses on
4678 APIs that output a list of labels and scores, as is the case for CVSs. Merge operations
4679 involve the mapping of multiple lists and associated scores, produced by multiple
4680 APIs, to just one list. For instance, a CVS receives a bowl of fruit as the input image
4681 and outputs the following:

4682 `[['apple', 0.9], ['banana', 0.8]]`

4683 where the first item is the label and the second item is the score. Similarly, another
4684 computer vision API outputs the following for the same image:

4685 `[[‘apple’, 0.7], [‘cherry’, 0.8]].`

4686 Merge operations can, therefore, merge these two responses into just one response.
4687 Naive ways of merging results could make use of *max*, *min*, and *average* operations
4688 on the confidence scores. For example, *max* merges results to:

4689 `[[‘apple’, 0.9], [‘banana’, 0.8], [‘cherry’, 0.8]];`

4690 *min* merges results to:

4691 `[[‘apple’, 0.7]];`

4692 and *average* merges results to:

4693 `[[‘apple’, 0.8], [‘banana’, 0.4], [‘cherry’, 0.4]].`

4694 However, as the object’s labels in each result are natural language, the operations
4695 do not exploit the label’s semantics when conducting label merging. To improve
4696 the quality of the merged results, we consider the ontologies of these labels, as we
4697 describe below.

4698 9.2.3 Merging Operators for Labels

4699 Merge operations on labels are n -ary operations that map R^n to R , where $R_i =$
4700 $\{(l_{ij}, s_{ij})\}$ is a response from endpoint i and contains pairs of labels (l_{ij}) and scores
4701 (s_{ij}). Merge operations on labels have the following properties:

- 4702 • *identity* defines that merging a single response should output same response
4703 (i.e., $R = \text{merge}(R)$ is always true);
- 4704 • *commutativity* defines that the order of operands should not change the result
4705 (i.e., $\text{merge}(R_1, R_2) = \text{merge}(R_2, R_1)$ is always true);
- 4706 • *reflexivity* defines that merging multiple same responses should output same
4707 response (i.e., $R = \text{merge}(R, R)$ is always true); and,
- 4708 • *additivity* defines that, for a specific label, the merged response should have
4709 higher or equal score for the label if a concrete endpoint has a higher score.
4710 Let $R = \text{merge}(R_1, R_2)$ and $R' = \text{merge}(R'_1, R_2)$ be merged responses. R_1 and
4711 R'_1 are same, except R'_1 has a higher score for label l_x than R_1 . The additive
4712 score property requires that R' score for l_x should be greater than or equal to
4713 R score for l_x .

4714 The *max*, *min*, and *average* operations in Section 9.2.2 follow each of these rules
4715 as all operations calculate the score by applying these operations on each score.

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

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

4716 9.3 Graph of Labels

4717 CVSs typically return lists of labels and their associated scores. In most cases, the
 4718 label can be a singular word (e.g., ‘husky’) or multiple words (e.g., ‘dog breed’).
 4719 Lexical databases, such as WordNet [244], can therefore be used to describe the
 4720 ontology behind these labels’ meanings. Figure 9.2 is an example of a graph of
 4721 labels and synsets. A synset is a grouped set of synonyms for a word. In this image,
 4722 we consider two fictional endpoints, endpoints 1–2. We label red nodes as labels
 4723 from endpoint 1, yellow nodes as labels from endpoint 2, and blue nodes as synsets
 4724 for the associated labels from both endpoints. As actual graphs are usually more
 4725 complex, Figure 9.2 is a simplified graph to illustrate the usage of associating labels
 4726 from two concrete sources to synsets.

4727 9.3.1 Labels and synsets

4728 The number of labels depends on input images and concrete API endpoints used.
 4729 Table 9.1 and Figure 9.3 show how many labels are returned, on average per image,
 4730 from Google Cloud Vision [422], Amazon Rekognition [397] and Azure Computer
 4731 Vision [436] image recognition APIs. These statistics were calculated using 1,000
 4732 images from Open Images Dataset V4 [424] Image-Level Labels set.

4733 Labels from Amazon and Microsoft tend to have corresponding synsets, and
 4734 therefore these endpoints return common words that are found in WordNet. On the
 4735 other hand, Google’s labels have less corresponding synsets: for example, labels
 4736 without corresponding synsets are car models and dog breeds.¹

4737 9.3.2 Connected Components

4738 A connected component (CC) is a subgraph in which there are paths between any
 4739 two nodes. In graphs of labels and synsets, CCs are clusters of labels and synsets
 4740 with similar semantic meaning. For instance, there are two CCs in Figure 9.2. CC 1
 4741 in Figure 9.2 has ‘beverage’, ‘dessert’, ‘chocolate’, ‘hot chocolate’,
 4742 ‘drink’, and ‘food’ labels from the red first endpoint and ‘coffee’, ‘hot
 4743 chocolate’, ‘drink’, ‘caffeine’, and ‘tea’ labels from the yellow second
 4744 endpoint. Therefore, these labels are related to ‘drink’. On the other hand, CC 2
 4745 in Figure 9.2 has ‘cup’ and ‘coffee cup’ labels from the first red endpoint and
 4746 ‘cup’, ‘coffee cup’, and ‘tableware’ labels from the yellow second endpoint.
 4747 These labels are, therefore, related to ‘cup’.

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

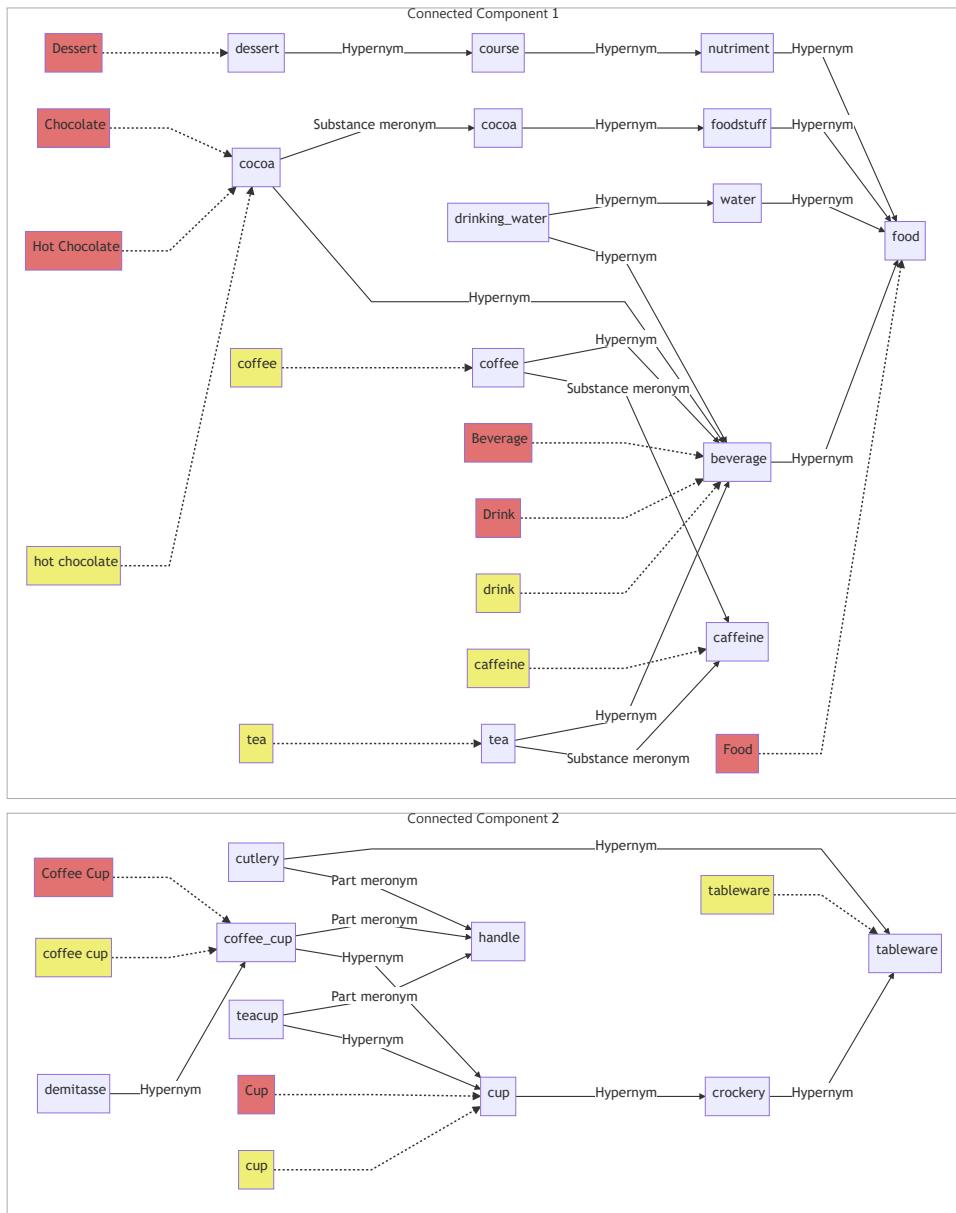


Figure 9.2: Graph of labels from two concrete endpoints (red and yellow) and their associated synsets related to both words (blue).

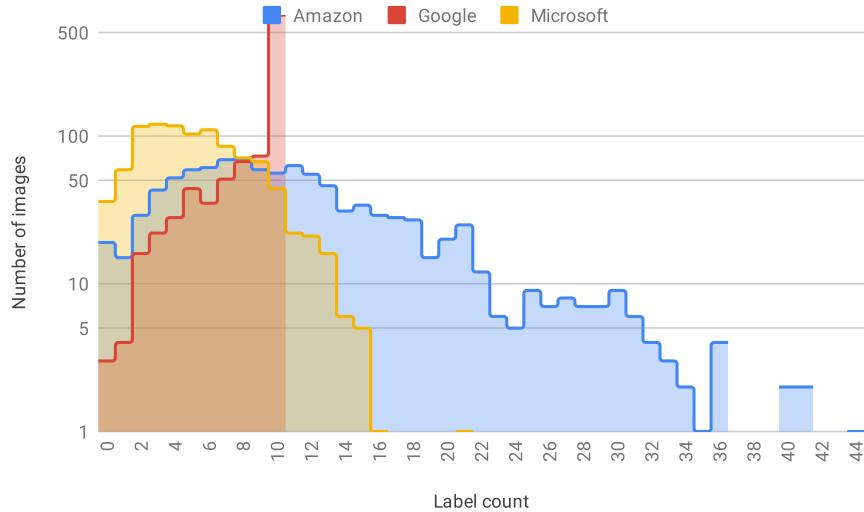


Figure 9.3: Number of labels responded from our input dataset to three concrete APIs assessed.

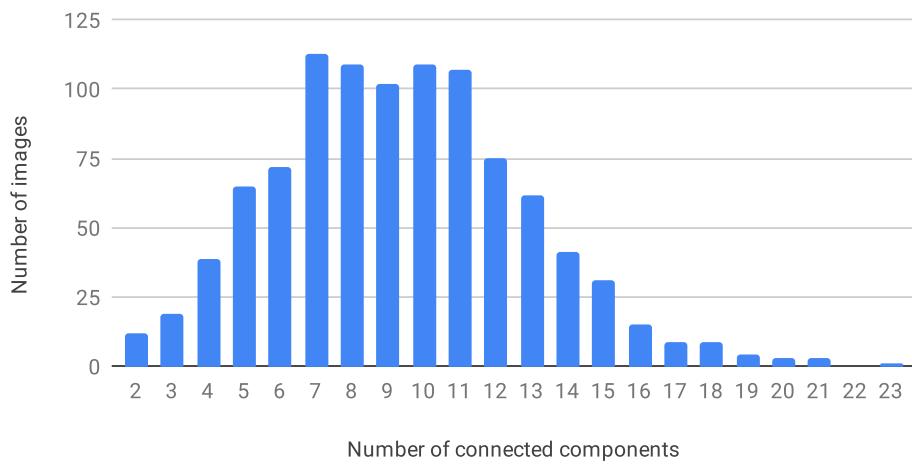


Figure 9.4: Number of connected components compared to the number of images.

4748 Figure 9.4 shows a distribution of number of CCs for the 1,000-image label
4749 detections on Amazon Rekognition, Google Cloud Vision, and Azure Computer
4750 Vision APIs. The average number of CCs is 9.36 ± 3.49 . The smaller number of
4751 CCs means that most of labels have similar meanings, while a larger value means
4752 that the labels are more disparate.

4753 9.4 API Results Merging Algorithm

4754 Our proposed algorithm to merge labels consists of four parts: (1) mapping labels to
4755 synsets, (2) deciding the total number of labels, (3) allocating the number of labels
4756 to CCs, and (4) selecting labels from CCs.

4757 9.4.1 Mapping Labels to Synsets

4758 Labels returned in CVS responses are words (in natural language) that do not always
4759 identify their intended meanings. For instance, a label *orange* may represent the
4760 fruit, the colour, or the name of the longest river in South Africa. To identify the
4761 actual meanings behind a label, our facade enumerates all synsets corresponding to
4762 labels. It then finds the most likely synsets for labels by traversing WordNet links.
4763 For instance, if an API endpoint outputs the ‘orange’ and ‘lemon’ labels, the
4764 facade regards ‘orange’ as a related synset word of ‘fruit’. If an API endpoint
4765 outputs ‘orange’ and ‘water’ labels, the facade regards ‘orange’ as a ‘river’.

4766 9.4.2 Deciding Total Number of Labels

4767 The number of labels in responses from endpoints vary as described in Section 9.3.1.
4768 The facade decides the number of merged labels using the numbers of labels from
4769 each endpoint. We formulate the following equation to calculate the number of
4770 labels:

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

4771 where $|R|$ is number of labels and scores in response, and n is number of endpoints.
4772 In case of naive operations in Section 9.2.2, the following is true:

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

4773 The proposal uses $\lfloor \sum_i|R_i|/n \rfloor$ to conform to the necessary condition described in
4774 Section 9.4.3.

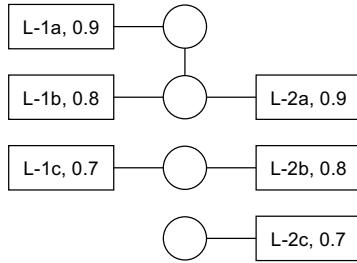


Figure 9.5: Allocation to connected components.

9.4.3 Allocating Number of Labels to Connected Components

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

Allegorically, allocating the number of labels to CCs is similar to proportional representation in a political voting system, where CCs are the political parties and labels are the votes to a party. Several allocation algorithms are introduced in proportional representation, for instance, the D’Hondt and Hare-Niemeyer methods [259]. However, there are differences from proportional representation in the political context. For label merging, labels have scores and origin endpoints and such information may improve the allocation algorithm. For instance, CCs supported with more endpoints should have a higher allocation than CCs with fewer endpoints, and CCs with higher scores should have a higher allocation than CCs with lower scores. We introduce an algorithm to allocate the number of labels to CCs. This allocates more to a CC with more supporting endpoints and higher scores. The steps of the algorithm are:

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

Tables 9.2 to 9.5 are examples of allocation iterations. In Table 9.2, the facade sorts scores separately for each endpoint. For instance, the first CC in Figure 9.5 has scores of 0.9 and 0.8 from endpoint 1 and 0.9 from endpoint 2. All CCs have a

Table 9.2: Allocation iteration 1.

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

Table 9.4: Allocation iteration 3.

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

Table 9.3: Allocation iteration 2.

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

Table 9.5: Allocation iteration 4.

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

4807 non-empty score array or more, so the facade skips Step II. The facade then picks
 4808 the highest scores for each endpoint and CC. CC 1 has the largest product of highest
 4809 scores and receives an allocation. In Table 9.3, the first CC removes every first score
 4810 in its array as it received an allocation in Table 9.2. In this iteration, the second CC
 4811 has largest product of scores and receives an allocation. In Table 9.4, the second CC
 4812 removes every first score in its array. At Step II, all the three CCs have an empty
 4813 array. The facade removes one empty array from each CC. In Table 9.5, the first CC
 4814 receives an allocation. The algorithm is applicable if total number of allocation is
 4815 less than or equal to $\max_i(|R_i|)$ as scores are removed in Step II. The condition is a
 4816 necessary condition.

4817 9.4.4 Selecting Labels from Connected Components

4818 For each CC, the facade applies the *average* operator from Section 9.2.2 and takes
 4819 labels with n -highest scores up to allocation, as per Section 9.4.3.

4820 9.4.5 Conformance to properties

4821 Section 9.2.3 defines four properties: identity, commutativity, reflexivity, and addi-
 4822 tivity. Our proposed method conforms to these properties:

- 4823 • *identity*: the method outputs same result if there is one response;
- 4824 • *commutativity*: the method does not care about ordering of operands;
- 4825 • *reflexivity*: the allocations to CCs are same to number of labels in CCs; and
- 4826 • *additivity*: increases in score increases or does not change the allocation to
 4827 the corresponding CC.

4828 9.5 Evaluation

4829 9.5.1 Evaluation Method

4830 To evaluate the merge methods, we merged CVS results from three representative
 4831 image analysis API endpoints and compared these merged results against human-

4832 verified labels. Images and human-verified labels are sourced from 1,000 randomly-
 4833 sampled images from the Open Images Dataset V4 [424] Image-Level Labels test
 4834 set.

4835 The first three rows in Table 9.7 are the evaluation of original responses from
 4836 each API endpoint. Precision, recall, and F-measure in Table 9.7 do not reflect
 4837 actual values: for instance, it appears that Google performs best at first glance, but
 4838 this is mainly because Google’s labels are similar to that of the Open Images label
 4839 set.

4840 The Open Images Dataset uses 19,995 classes for labelling. The human-verified
 4841 labels for the 1,000 images contain 8,878 of these classes. Table 9.6 shows the
 4842 correspondence between each service’s labels and the Open Images Dataset classes.
 4843 For instance, Amazon Rekognition outputs 11,416 labels in total for 1,000 images.
 4844 There are 1,409 unique labels in 11,416 labels. 1,111 labels out of 1,409 can be
 4845 found in Open Images Dataset classes. Rekognition’s labels matches to Open Images
 4846 Dataset classes at 78.9% ratio, while Google has an outstanding matched percentage
 4847 of 94.1%. This high match is likely due to Google providing both Google Cloud
 4848 Vision and the Open Images Dataset—it is likely that they are trained on the same
 4849 data and labels. An endpoint with higher matched percentage has a more similar
 4850 label set to the Open Images Dataset classes. However, a higher matched percentage
 4851 does not mean imply *better quality* of an API endpoint; it will increase apparent
 4852 precision, recall, and F-measure only.

4853 The true and false positive (TP/FP) label averages and the TP/FP ratio is shown
 4854 in Table 9.7. Where the TP/FP ratio is larger, the scores are more reliable, however
 4855 it is possible to increase the TP/FP ratio by adding more false labels with low scores.
 4856 On the other hand, it is impossible to increase F-measure intentionally, because
 4857 increasing precision will decrease recall, and vice versa. Hence, the importance of
 4858 the F-measure statistic is critical for our analysis.

4859 Let R_A , R_G , and R_M be responses from Amazon Rekognition, Google Cloud
 4860 Vision, and Microsoft’s Azure Computer Vision, respectively. There are four sets
 4861 of operands, i.e., (R_A, R_G) , (R_G, R_M) , (R_M, R_A) , and (R_A, R_G, R_M) . Table 9.7
 4862 shows the evaluation of each operands set, Table 9.8 shows the averages of the four
 4863 operands sets, and Figure 9.6 shows the comparison of F-measure for each methods.

4864 9.5.2 Naive Operators

4865 Results of *min*, *max*, and *average* operators are shown in Tables 9.7 and 9.8 and Fig-
 4866 ure 9.6. The *min* operator is similar to *union* operator of set operation, and outputs
 4867 all labels of operands. The precision of the *min* operator is always greater than any
 4868 precision of operands, and the recall is always lesser than any precision of operands.
 4869 *Max* and *average* operators are similar to *intersection* operator of set operations.
 4870 Both operators output intersection of labels of operands and there is no clear relation
 4871 to the precision and recall of operands. Since both operators have the same preci-
 4872 sion, recall, and F-measure, Figure 9.6 groups them into one. The *average* operator
 4873 performs well on the TP/FP ratio, where most of the same labels from multiple
 4874 endpoints are TPs. In many cases of the four operand sets, all naive operators’

Table 9.6: Matching to human-verified labels.

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

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

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

Table 9.8: Average of the evaluation result.

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

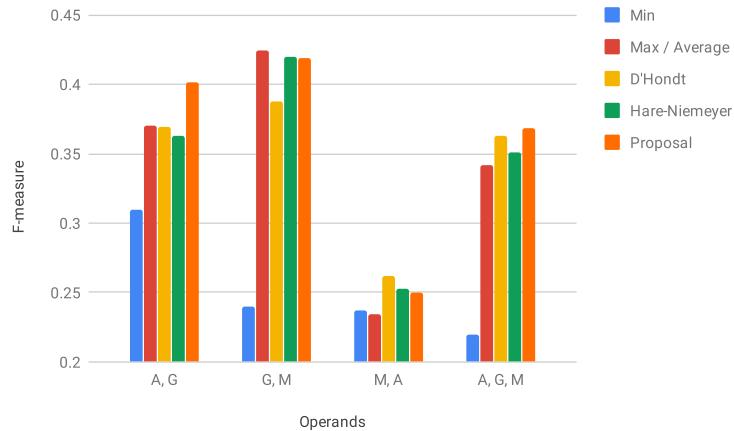


Figure 9.6: F-measure comparison.

4875 F-measures are between F-measures of operands. None of naive operators therefore
 4876 improve results by merging responses from multiple endpoints.

4877 **9.5.3 Traditional Proportional Representation Operators**

4878 There are many existing allocation algorithms in proportional representation, e.g.,
 4879 the Niemeyer and Niemeyer method [259]. These methods may be replacements of
 4880 those in Section 9.4.3. Other steps, i.e., Sections 9.4.1, 9.4.2 and 9.4.4, are the same
 4881 as for our proposed technique. Tables 9.7 and 9.8 and Figure 9.6 show the result of
 4882 these traditional proportional representation algorithms. Averages of F-measures by
 4883 traditional proportional representation operators are almost equal to that of the *max*
 4884 and *average* operators. It is worth noting that merging *M* and *A* responses results in
 4885 a better F-measure than each F-measure of *M* and *A* individually. As these are not
 4886 biased to human-verified labels, situations in the real-world usage should, therefore,
 4887 be similar to the case of *M* and *A*. Hence, RQ1 is true.

4888 **9.5.4 New Proposed Label Merge Technique**

4889 As shown in Table 9.8, our proposed new method performs best in F-measure.
 4890 Instead, the TP/FP ratio is less than *average*, the D'Hondt method, and Hare-
 4891 Niemeyer method. As described in Section 9.5.1, we argue that F-measure is a
 4892 more important measure than the TP/FP ratio (in this case). Therefore, RQ2 is
 4893 true. Shown in Table 9.7, our proposed new method improves the results when
 4894 merging *M* and *A* in non-biased endpoints. It is similar to traditional proportional
 4895 representation operators, but does not perform as well. However, it performs better
 4896 on other operand sets, and performs best overall as shown in Figure 9.6.

4897 **9.5.5 Performance**

4898 We used AWS EC2 m5.large instance (2 vCPUs, 2.5 GHz Intel Xeon, 8 GiB RAM);
 4899 Amazon Linux 2 AMI (HVM), SSD Volume Type; Node.js 8.12.0. It takes 0.370

4900 seconds to merge responses from three endpoints. Computational complexity of the
4901 algorithm in Section 9.4.3 is $O(n^2)$, where n is total number of labels in responses.
4902 (The estimation assumes that the number of endpoints is a constant.) Complexity of
4903 Step I in Section 9.4.3 is $O(n \log n)$, as the worst case is that all n labels are from
4904 one single endpoint and all n labels are in one CC. Complexity of Step II to Step V
4905 is $O(n^2)$, as the number of CCs is less than or equal to n and number of iterations
4906 are less than or equal to n . As Table 9.1 shows, the averaged total number of three
4907 endpoints is 25.58. Most of time for merging is consumed by looking up WordNet
4908 synsets (Section 9.4.1). The API facade calls each APIs on actual endpoints in
4909 parallel. It takes about 5 seconds, which is much longer than 0.370 seconds taken
4910 for the merging of responses.

4911 9.6 Conclusions and Future Work

4912 In this paper, we propose a method to merge responses from CVSs. Our method
4913 merges API responses better than naive operators and other proportional represen-
4914 tation methods (i.e., D'Hondt and Hare-Niemeyer). The average of F-measure of
4915 our method marks 0.360; the next best method, Hare-Niemeyer, marks 0.347. Our
4916 method and other proportional representation methods are able to improve the F-
4917 measure from original responses in some cases. Merging non-biased responses
4918 results in an F-measure of 0.250, while original responses have an F-measure be-
4919 tween 0.246 and 0.242. Therefore, users can improve their applications' precision
4920 with small modification, i.e., by switching from a singular URL endpoint to a facade-
4921 based architecture. The performance impact by applying facades is small, because
4922 overhead in facades is much smaller than API invocation. Our proposal method
4923 conforms identity, commutativity, reflexivity, and additivity properties and these
4924 properties are advisable for integrating multiple responses.

4925 Our idea of a proportional representation approach can be applied to other IWSs.
4926 If the response of such a service is list consisting of an entity and score, and if there is a
4927 way to group entities, a proposal algorithm can be applied. The opposite approach is
4928 to improve results by inferring labels. Our current approach picks some of the labels
4929 returned by endpoints. IWSs are not only based on supervised ML—thus to cover a
4930 wide range of IWSs, it is necessary to classify and analyse each APIs and establish
4931 a method to improve results by merging. Currently graph structures of labels and
4932 synsets (Figure 9.2) are not considered when merging results. Propagating scores
4933 from labels could be used, losing the additivity property but improving results for
4934 users. There are many ways to propagate scores. For instance, setting propagation
4935 factors for each link type would improve merging and could be customised for users'
4936 preferences. It would be possible to generate an API facade automatically. APIs
4937 with the same functionality have same or similar signatures. Machine-readable API
4938 documentation, for instance, OpenAPI Specification, could help a generator to build
4939 an API facade.

CHAPTER 10

4940

4941

4942

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

4943

4944 **Abstract** Increased popularity of ‘intelligent’ web services provides end-users with machine-
4945 learnt functionality at little effort to developers. However, these services require a decision
4946 threshold to be set which is dependent on problem-specific data. Developers lack a systematic
4947 approach for evaluating intelligent services and existing evaluation tools are predominantly
4948 targeted at data scientists for pre-development evaluation. This paper presents a workflow
4949 and supporting tool, Threshy, to help *software developers* select a decision threshold suited to
4950 their problem domain. Unlike existing tools, Threshy is designed to operate in multiple work-
4951 flows including pre-development, pre-release, and support. Threshy is designed for tuning
4952 the confidence scores returned by intelligent web services and does not deal with hyper-
4953 parameter optimisation used in ML models. Additionally, it considers the financial impacts
4954 of false positives. Threshold configuration files exported by Threshy can be integrated into
4955 client applications and monitoring infrastructure. Demo: <https://bit.ly/2YKeYhE>.

4956 10.1 Introduction

4957 Machine learning algorithm adoption is increasing in modern software. End users
4958 routinely benefit from machine-learnt functionality through personalised recom-
4959 mendations [82], voice-user interfaces [254], and intelligent digital assistants [52]. The
4960 easy accessibility and availability of intelligent web services (IWSs)¹ is contributing
4961 to their adoption. These IWSs simplify the development of machine learning (ML)

[†]This chapter is originally based on A. Cummaudo, S. Barnett, R. Vasa, and J. Grundy, “Threshy: Supporting Safe Usage of Intelligent Web Services,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event, USA: ACM, November 2020. DOI 10.1145/3368089.3417919, pp. 1645–1649. Terminology has been updated to fit this thesis.

¹Such as Azure Computer Vision (<https://azure.microsoft.com/en-au/services/cognitive-services/computer-vision/>), Google Cloud Vision (<https://cloud.google.com/vision/>), or Amazon Rekognition (<https://aws.amazon.com/rekognition/>).

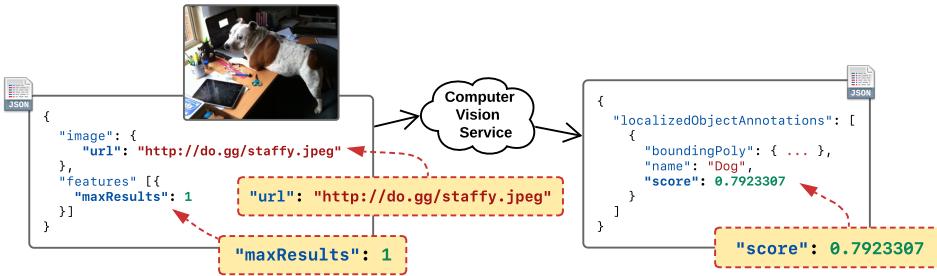


Figure 10.1: Request and response for an intelligent computer vision web service with only three configuration parameters: the image’s url, maxResults and score.

4962 solutions as they (i) do not require specialised ML expertise to build and maintain,
 4963 (ii) abstract away infrastructure related issues associated with ML [16, 320], and
 4964 (iii) provide web application programming interfaces (APIs) for ease of integration.

4965 However, unlike traditional web services, the functionality of these IWSs is
 4966 dependent on a set of assumptions unique to ML [88]. These assumptions are based
 4967 on the data used to train ML algorithms, the choice of algorithm, and the choice of
 4968 data processing steps—most of which are not documented. For developers, these
 4969 assumptions mean that the performance characteristics of an IWS in any particular
 4970 application problem domain is not fully knowable. IWSs represent this uncertainty
 4971 through a confidence value associated with their predictions.

4972 As an example, consider Figure 10.1, which illustrates an image of a dog up-
 4973 loaded to a real computer vision service (CVS). Developers have very few configura-
 4974 tion parameters in the upload payload (`url` of the image to analyse and `maxResults`
 4975 the number of objects to detect). The JSON output payload returns the confidence
 4976 value via a `score` field (0.792), the bounding box and a “dog” label. Developers
 4977 can only work with these parameters; unlike hyper-parameter optimisation available
 4978 to ML creators, who can configure the internal parameters of the algorithm while
 4979 training a model. Given the structure of the abstractions, developers have no insight
 4980 into which hyper-parameters are used or the algorithm selected and cannot tune the
 4981 underlying trained model when using an IWS. Thus an evaluation procedure must
 4982 be followed as a part of using an IWS for an application to work with and tune the
 4983 output confidence values for a given input set.

4984 A typical evaluation process would involve a test data set (curated by the devel-
 4985 opers using the IWS) that is used to determine an appropriate threshold. Choice of
 4986 a decision threshold is a critical element of the evaluation procedure [150]. This is
 4987 especially true for classification problems such as detecting if an image contains can-
 4988 cer. Simple approaches to selecting a threshold are often insufficient, as highlighted
 4989 in Google’s ML course: *“It is tempting to assume that [a] classification threshold
 4990 should always be 0.5, but thresholds are problem-dependent, and are therefore
 4991 values that you must tune.”*²

4992 As an example consider the predictions from two email spam classifiers shown

²See <https://bit.ly/36oMgWb>.

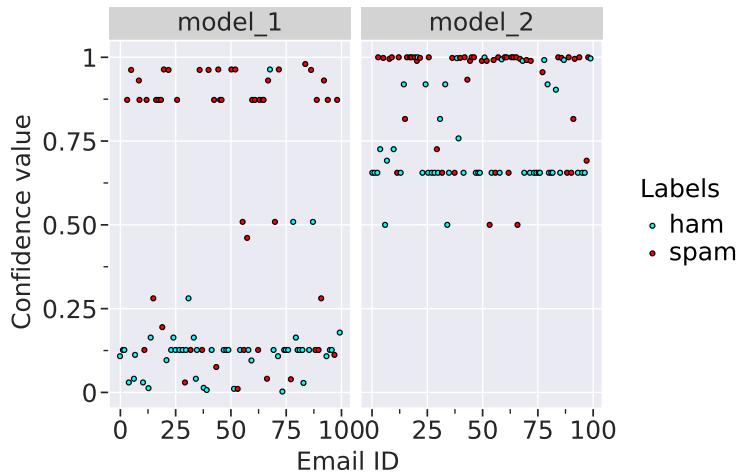


Figure 10.2: Predictions for 100 emails from two spam classifiers. Decision thresholds are classifier-dependent: a single threshold for both classifiers is *not* appropriate as ham emails are clustered at 0.12 (model_1) and at 0.65 (model_2). Developers must evaluate performance for *both* thresholds.

4993 in Figure 10.2. The predicted safe emails, ‘ham’, are in two separate clusters (a
 4994 simple threshold set to approx. 0.2 for model 1 and 0.65 for model 2, indicating
 4995 that different decision thresholds may be required depending on the classifier. Also
 4996 note that some emails have been misclassified; how many depends on the choice of
 4997 decision threshold. An appropriate threshold considers factors outside algorithmic
 4998 performance, such as financial cost and impact of wrong decisions. To select an
 4999 appropriate decision threshold, developers using intelligent services need approaches
 5000 to reason about and consider trade-offs between competing *cost factors*. These
 5001 include impact, financial costs, and maintenance implications. Without considering
 5002 these trade-offs, sub-optimal decision thresholds will be selected.

5003 The standard approach for tuning thresholds in classification problems involve
 5004 making trade-offs between the number of false positives and false negatives using
 5005 the receiver operating characteristic (ROC) curve. However, developers (i) need
 5006 to realise that this trade-off between false positives and false negatives is a data
 5007 dependent optimisation process [319], (ii) often need to develop custom scripts
 5008 and follow a trial-and-error based approach to determine a threshold, (iii) must
 5009 have appropriate statistical training and expertise, and (iv) be aware that multi-
 5010 label classification require more complex optimisation methods when setting label
 5011 specific costs. However, current intelligent services do not sufficiently guide or
 5012 support software engineers through the evaluation process, nor do they make this
 5013 need clear in the documentation.

5014 In this paper we present **Threshy**³, a tool to assist developers in selecting de-
 5015 cision thresholds when using intelligent services. The motivation for developing
 5016 Threshy arose from our work across a set of industry projects. Unlike existing
 5017 tooling (see Section 10.4), **Threshy serves as a means to up-skill and educate**

³Threshy is available for use at <http://bit.ly/a2i2-threshy>

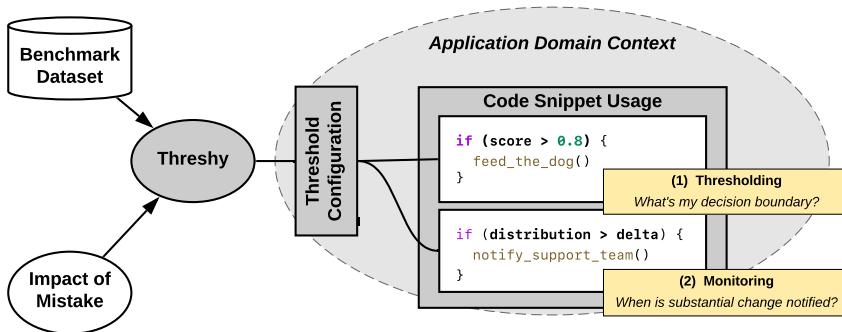


Figure 10.3: Threshy supports two key aspects for intelligent web services: threshold selection and monitoring.

5018 **software engineers in selecting machine-learnt decision thresholds**, for example,
 5019 on aspects such as confusion matrices. We re-iterate that the end-users of Threshy
 5020 are software engineers and not data scientists—Threshy is not designed for hyper-
 5021 parameter tuning of models, but for threshold tuning to use intelligent web services
 5022 more robustly where internal models are not exposed. Threshy provides a visually
 5023 interactive interface for developers to fine-tune thresholds and explore trade-offs of
 5024 prediction hits/misses. This exposes the need for optimisation of thresholds, which
 5025 is dependent on particular use cases.

5026 Threshy improves developer productivity through automation of the threshold
 5027 selection process by leveraging an optimisation algorithm to propose thresholds.
 5028 Figure 10.3 illustrates the two key aspects by which Threshy supports developer’s
 5029 application domain context. Developers input a representative dataset of their applica-
 5030 tion data (a benchmark dataset) in addition to cost factors to Threshy. Threshy’s
 5031 output helps developers select appropriate thresholds and can be used to monitor
 5032 the evolution of an IWS. This algorithm considers different cost factors providing
 5033 developers with summary information so they can make more informed trade-offs.
 5034 Developers also benefit from the workflow implemented in Threshy by providing a
 5035 reproducible procedure for testing and tuning thresholds for any category of clas-
 5036 sification problem (binary, multi-class, and multi-label). Threshy has also been
 5037 designed to work for different input data types including images, text and categor-
 5038 ical values. The output, is a configuration file that can be integrated into client
 5039 applications ensuring that the thresholds can be updated without code changes, and
 5040 continuously monitored in a production setting.

5041 10.2 Motivating Example

5042 As a motivating example consider Nina, a fictitious developer, who has been em-
 5043 ployed by Lucy’s Tomato Farm to automate the picking of tomatoes from their vines
 5044 (when ripe) using computer vision and a harvesting robot. Lucy’s Farm grow five
 5045 types of tomatoes (roma, cherry, plum, green, and yellow tomatoes). Nina’s robot—
 5046 using an attached camera—will crawl and take a photo of each vine to assess it for

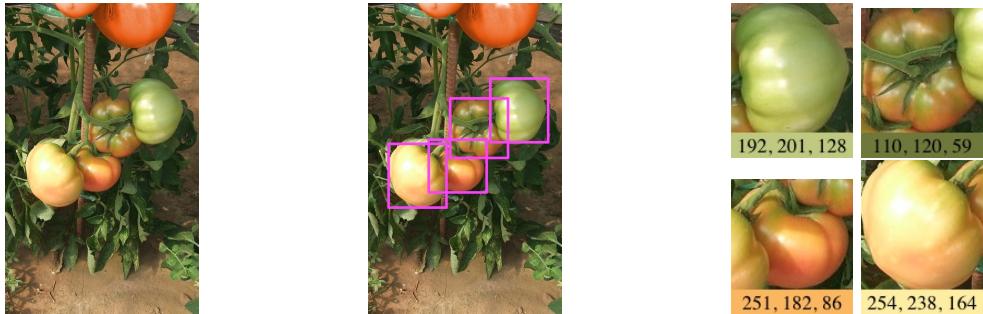


Figure 10.4: Pipeline of Nina’s harvesting robot. *Left:* Photo from harvesting robot’s webcam. *Centre:* Classification detecting different types of tomatoes. *Right:* Binary classification for ripeness (ripe/unripe) based on (R, G, B values).

5047 harvesting. Nina’s automated harvester needs to sort picked tomatoes into a respec-
 5048 tive container, and thus several business rules need to be encoded into the prediction
 5049 logic to sort each tomato detected based on its *ripeness* (ripe or not ripe) and *type of*
 5050 *tomato* (as above). Nina uses a two-stage pipeline consisting of a multi-class and a
 5051 binary classification model. She has decided to evaluate the viability of cloud based
 5052 intelligent services and use them if operationally effective.

5053 Figure 10.4 illustrates an example of the pipeline as listed below:

- 5054 1. **Classify tomato ‘type’.** This stage uses an object localisation service to detect
 5055 all tomato-like objects in the frame and classifies each tomato into one of the
 5056 following labels: [‘roma’, ‘cherry’, ‘plum’, ‘green’, ‘yellow’, ‘unknown’].
- 5057 2. **Assess tomato ‘ripeness’.** This stage uses a crop of the localised tomatoes
 5058 from the original frame to assess the crop’s colour properties (i.e., average
 5059 colour must have $R > 200$ and $G < 240$). This produces a binary classification
 5060 to deduce whether the tomato is ripe or not.

5061 Nina only has a minimal appreciation of the evaluation method to use for off-
 5062 the-shelf computer vision (classification) services. She also needs to consider the
 5063 financial costs of misclassifying either the tomato type or the ripeness. Missing a
 5064 few ripe tomatoes isn’t a significant concern as the robot travels the field twice a
 5065 week during harvest season. However, picking an unripe tomato is expensive as
 5066 Lucy cannot sell them. Therefore, Nina needs a better (automated) way to assess
 5067 the performance of the service and set optimal thresholds for her picking robot, to
 5068 maximise profit.

5069 To assist in developing Nina’s pipeline, Lucy sampled a section of 1000 tomatoes
 5070 by taking a photo of each tomato, manually labelling its type, and assessing whether
 5071 the vine was ‘ripe’ or ‘not_ripe’. Nina ran the labelled images through an IWS,
 5072 with each image having a predicted type (multi-class) and ripeness (binary), with
 5073 respective confidence values.

5074 Nina combined the predictions, their respective confidence values, and Lucy’s
 5075 labelled ground truths into a CSV file which was then uploaded to Threshy. Nina
 5076 asked Lucy the farmer to assist in setting relevant costs (from a business perspective)
 5077 for correct predictions and false predictions. Threshy then recommended a choice

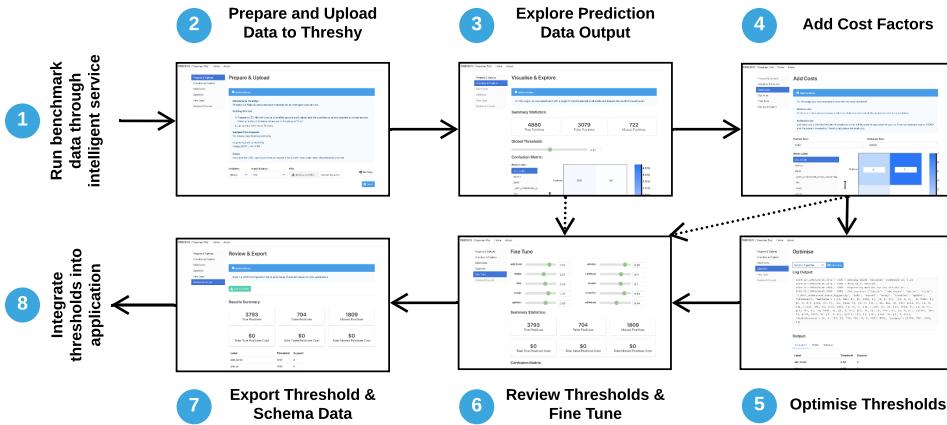


Figure 10.5: UI workflow for interacting with Threshy to optimise the thresholds for classification problem.

of decision threshold which Nina then fine tuned while considering the performance and cost implications.

10.3 Threshy

Threshy is a tool to assist software engineers with setting decision thresholds when integrating machine-learnt components in a system in collaboration with subject matter experts. Our tool also serves as a method to inform and educate engineers about the nuances to consider when using prepackaged ML services. Key novel features are:

- Automating threshold selection using an optimisation algorithm (NSGA-II [96]), optimising the results for each label.
- Support for user defined, domain-specific weights when optimising thresholds, such as financial costs and impact to society. This allows decision thresholds to be set within a business context as they differ between applications [107].
- Handles nuances of classification problems such as dealing with multi-objective optimisation, and metric selection—reducing errors of omission.
- Support key classification problems including binary (e.g. email is spam or ham), multi-class (e.g. predict the colour of a car), and multi-label (e.g. assign multiple topics to a document). Existing tools ignore multi-label classification.

Setting thresholds in Threshy is an eight step process as outlined in Figure 10.5. Software engineers ① run a benchmark dataset through the machine-learnt component to create a data file (CSV format) with true labels and predicted labels along with the predicted confidence values. The data file is then ② uploaded for initial exploration where engineers can ③ experiment with modifying a single global threshold for the dataset. Developers may choose to exit at this point (as indicated by dotted arrows in Figure 10.5). Optionally, the engineer ④ defines costs for missed predictions followed by selecting optimisation settings. The optional optimisation

5104 step of Threshy ⑤ considers the performance and costs when deriving the thresh-
5105 olds. Finally, the engineer can ⑥ review and fine tune the calculated thresholds,
5106 associated costs, and ⑦ download generated threshold meta-data to be ⑧ integrated
5107 into their application.

5108 Threshy runs a client/server architecture with a thin-client (see Figure 10.6).
5109 The web-based application consists of an interactive front-end where developers
5110 upload benchmark results—consisting of both human annotated labels and machine
5111 predictions from the IWS—and use threshold tuners (via sliders) to present a data
5112 summary of the uploaded information. Predicted model performances and costs are
5113 entered manually into the web interface by the developer. The Threshy back-end
5114 runs a data analyser, cost processor and metrics calculator when relevant changes
5115 are made to the front-end’s tuning sliders. Separating the two concerns allows for
5116 high intensity processing to be done on the server and not the front end.

5117 The data analyser provides a comprehensive overview of confusion matrices
5118 compatible for multi-label multi-class classification problems. When representing
5119 the confusion matrix, it is trivial to represent instances where multi-label multi-
5120 classification is not considered. For example, in the simplest case, a single row in
5121 the matrix represents a single label out of two classes, or each row has one label but it
5122 has multiple classes. However, a more challenging case to visualise arises when you
5123 have n labels and n classes; the true/false matches become too excessive to visualise
5124 as it is disproportionate to the true results. To deal with this issue, we condense the
5125 summary statistics down to three constructs: (i) number of true positives, (ii) false
5126 positives, (iii) missed positives. This allows us to optimise against the true positives
5127 and minimise the other two constructs. Threshy is a fully self-contained repository
5128 of the tool implementation, scripting and exploratory notebooks, which is available
5129 at <https://github.com/a2i2/threshy>.

5130 10.4 Related work

5131 Optimal machine-learnt decision boundaries depend on identifying the operating
5132 conditions of the problem domain. A systematic study by Drummond and Holte
5133 [107] classifies four operating conditions to determine a decision threshold: (i) the
5134 operating condition is known and the model trained matches perfectly; (ii) where
5135 the operating conditions are known but change with time, and thus the model must
5136 be adaptable to such changes; (iii) where there is uncertainty in the knowledge of
5137 the operating conditions certain changes in the operating condition are more likely
5138 than others; (iv) where there is no knowledge of the operating conditions and the
5139 conditions may change from the model in any possible way. Various approaches
5140 to determine appropriate thresholds exist for all four of these cases, such as cost-
5141 sensitive learning, ROC analysis, and Brier scores. However, an *automated* attempt
5142 to calibrate decision threshold boundaries is not considered, and is largely pitched
5143 at a non-software engineering audience. A recent study touches on this in model
5144 management for large-scale adversarial instances in Google’s advertising system
5145 [319], however this is only a single component within the entire architecture, and is
5146 not a tool that is useful for developer’s in varying contexts. Our Threshy provides a

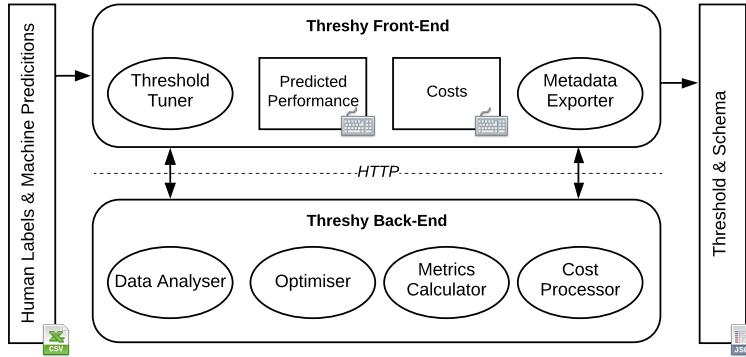


Figure 10.6: Architecture of Threshy.

‘plug-and-play’ style calibration method where any context/domain can have thresholds automatically calibrated *and* optimised for engineers. Threshy’s architecture supports a headless mode for use in monitoring workflows.

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 [160, 276]. However, a recent emphasis to probe only inputs and outputs of a model has been explored, exploring off-the-shelf models without knowledge of its unknowns (see Figure 10.2) to reflect the nature of real-world development. Google’s *What-If Tool* [376] 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 [159] provides an interface to test hypotheticals on Generalized Additive Models, and a *ModelTracker* tool [13] collates summary statistics on sample data to enable visualisation of model behaviour and access to key performance metrics.

However, these tools are focused toward pre-development model evaluation and not designed for software engineering workflows. Nor are they context-aware to the overall software system they are meant to target. They are also aimed at data scientists and model builders and do not consider consistent tooling that works across development, test, and production environments. Further, certain tools are tied to specific ML frameworks (e.g., What-If and Tensorflow). Our work, instead, attempts to bridge these gaps through a context-aware, structured workflow with an automated tool targeted to software developers; our tool is designed for software engineers to calibrate their thresholds and is used for IWS APIs in particular.

10.5 Conclusions & Future Work

Primary contributions of this work include Threshy, a tool for automating threshold selection, and the overall meta-workflow proposed in Threshy that developers can use as a point of reference for calibrating thresholds. In future work, we plan to evaluate Threshy with software engineers to identify additional insights required to

- 5176 make decision thresholds in practice and add code synthesis for monitoring concept drift and for implementing decision thresholds.
- 5177

5178

5179

5180 An Integration Architecture Tactic to guard AI-first Components[†]

5181

5182 **Abstract** Intelligent web services provide the power of AI to developers via simple REST-
5183 ful API endpoints, abstracting away many complexities of machine learning. However,
5184 most of these intelligent web services (IWSs)—such as computer vision—continually learn
5185 with time. When the internals within the abstracted ‘black box’ become hidden and evolve,
5186 pitfalls emerge in the robustness of applications that depend on these evolving services.
5187 Without adapting the way developers plan and construct projects reliant on IWSs, signifi-
5188 cant gaps and risks result in both project planning and development. Therefore, how can
5189 software engineers best mitigate software evolution risk moving forward, thereby ensuring
5190 that their own applications maintain quality? Our proposal is an architectural tactic designed
5191 to improve intelligent service-dependent software robustness. The tactic involves creating
5192 an application-specific benchmark dataset baselined against an intelligent service, enabling
5193 evolutionary behaviour changes to be mitigated. A technical evaluation of our implemen-
5194 tation of this architecture demonstrates how the tactic can identify 1,054 cases of substantial
5195 confidence evolution and 2,461 cases of substantial changes to response label sets using a
5196 dataset consisting of 331 images that evolve when sent to a service.

5197 11.1 Introduction

5198 The introduction of intelligent web services (IWSs) into the software engineering
5199 ecosystem allows developers to leverage the power of artificial intelligence (AI)
5200 without implementing complex AI algorithms, source and label training data, or
5201 orchestrate powerful and large-scale hardware infrastructure. This is extremely

[†]This chapter is originally based on A. Cummaudo, S. Barnett, R. Vasa, J. Grundy, and M. Abd-elrazek, “Beware the evolving ‘intelligent’ web service! An integration architecture tactic to guard AI-first components,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event, USA: ACM, November 2020. DOI 10.1145/3368089.3409688, pp. 269–280. Terminology has been updated to fit this thesis.

5202 enticing for developers to embrace due to the effort, cost and non-trivial expertise
5203 required to implement AI in practice [287, 320].

5204 However, the vendors that offer these services also periodically update their
5205 behaviour (responses). The ideal practice for communicating the evolution of a
5206 web service involves updating the version number and writing release notes. The
5207 release notes typically describe new capabilities, known problems, and requirements
5208 for proper operation [50]. Developers anticipate changes in behaviour between ver-
5209 sioned releases although they expect the behaviour of a specific version to remain
5210 stable over time [362]. However, emerging evidence indicates that ‘intelligent’ ser-
5211 vices *do not* communicate changes explicitly [87]. Intelligent services evolve in
5212 unpredictable ways, provide no notification to developers and changes are undocu-
5213 mented [91]. To illustrate this, consider Figure 11.1, which shows the evolution of a
5214 popular computer vision service (CVS) with examples of labels and associated confi-
5215 dence scores changing are shown. This behaviour change severely negatively affects
5216 reliability. Applications may no longer function correctly if labels are removed or
5217 confidence scores change beyond predefined thresholds.

5218 Unlike traditional web services, the functionality of these IWSs is dependent
5219 on a set of assumptions unique to their machine learning principles and algorithms.
5220 These assumptions are based on the data used to train machine learning algorithms,
5221 the choice of algorithm, and the choice of data processing steps—most of which
5222 are not documented to service end users. The behaviour of these services evolve
5223 over time [88]—typically this implies the underlying model has been updated or
5224 re-trained.

5225 Vendors do not provide any guidance on how best to deal with this evolution in
5226 client applications. For developers to discover the impact on their applications they
5227 need to know the behavioural deviation and the associated impact on the robustness
5228 and reliability of their system. Currently, there is no guidance on how to deal with
5229 this evolution, nor do developers have an explicit checklist of the likely errors and
5230 changes that they must test for [91].

5231 In this paper, we present a reference architecture to detect the evolution of such
5232 IWSs, using a mature subset of these services that provide computer vision as an
5233 exemplar. This tactic can be used both by intelligent service consumers, to defend
5234 their applications against the evolutionary issues present in IWSs, and by service
5235 vendors to make their services more robust. We also present a set of error conditions
5236 that occur in existing CVSs.

5237 The key contributions of this paper are:

- 5238 • A set of new service error codes for describing the empirically observed error
5239 conditions in IWSs.
- 5240 • A new reference architecture for using IWSs with a Proxy Server that returns
5241 error codes based on an application specific benchmark dataset.
- 5242 • A labelled data set of evolutionary patterns in CVSs.
- 5243 • An evaluation of the new architecture and tactic showing its efficacy for
5244 supporting IWS evolution from both provider and consumer perspectives.

5245 The rest of this paper is organised thus: Section 11.2 presents a motivating



'natural foods' (.956) → 'granny smith' (.986)



'skiing' (.937) → 'snow' (.982)



'girl' (.660) → 'photography' (.738)



'water' (.972) → 'wave' (.932)



'tennis' (.982) → 'sports' (.989)



'neighbourhood' (.925) → 'blue' (.927)

Figure 11.1: Prominent CVSSs evolve with time which is not effectively communicated to developers. Each image was uploaded in November 2018 and March 2019 and the topmost label was captured. Specialisation in labels (*Left*), generalisation in labels (*Centre*) and emphasis change in labels (*Right*) are all demonstrated from the same service with no API change and limited release note documentation. Confidence values indicated in parentheses.

example that anchors our work; Section 11.3 presents a landscape analysis on IWSs; Section 11.4 presents an overview of our architecture; Section 11.5 describes the technical evaluation; Section 11.6 presents a discussion into the implications of our architecture, its limitations and potential future work; Section 11.7 discusses related work; Section 11.8 provides concluding remarks.

5251 11.2 Motivating Example

5252 We identify the key requirements for managing evolution of IWSs using a motivating
5253 example. Consider Michelina, a software engineer tasked with developing a fall
5254 detector system for helping aged care facilities respond to falls promptly. Michelina
5255 decides to build the fall detector with an intelligent service for detecting people as she
5256 has no prior experience with machine learning. The initial system built by Michelina
5257 consists of a person detector and custom logic to identify a fall based on rapid shape
5258 deformation (i.e., a vertical ‘person’ changing to a horizontal ‘person’ greater than
5259 specified probability threshold value). Due to the inherent uncertainty present in
5260 an intelligent service and the importance of correctly identifying falls, Michelina
5261 informs the aged care facility that they should manually verify falls before dispatching
5262 a nurse to the location. The aged care facility is happy with this approach but inform
5263 Michelina that only a certain percentage of falls can be manually verified based on
5264 the availability of staff. In order to reduce the manual work Michelina sets thresholds
5265 for a range of confidence scores where the system is uncertain. Michelina completes
5266 the fall detector using a well-known cloud-based intelligent image classification web
5267 service and her client deploys this new fall detection application.

5268 Three months go by and then the aged care facility contact Michelina saying the
5269 percentage of manual inspections is far too high and could she fix it. Michelina is
5270 mystified why this is occurring as she thoroughly tested the application with a large
5271 dataset provided by the aged care facility. On further inspection Michelina notices
5272 that the problem is caused by some images classifying the person with a ‘child’
5273 label rather than a ‘person’ label. Michelina is frustrated and annoyed at this
5274 behaviour as (i) the cloud vendor did not document or notify her of the change of the
5275 intelligent service behaviour, (ii) she does not know the best practice for dealing with
5276 such a service evolution, and (iii) she cannot predict how the service will change
5277 in the future. This experience also makes Michelina wonder what other types of
5278 evolution can occur and how can she minimise these behavioural changes on her
5279 critical care application. Michelina then begins building an ad-hoc solution hoping
5280 that what she designs will be sufficient.

5281 For Michelina to build a robust solution she needs to support the following
5282 requirements:

- 5283 **R1.** Define a set of error conditions that specify the types of evolution that occur
5284 for an intelligent service.
- 5285 **R2.** Provide a notification mechanism for informing client applications of be-
5286 havioural changes to ensure the robustness and reliability of the application.

- 5287 **R3.** Monitor the evolution of IWSs for changes that affect the application's be-
5288 haviour.
- 5289 **R4.** Implement a flexible architecture that is adaptable to different IWSs and ap-
5290 plication contexts to facilitate reuse.

5291 **11.3 Intelligent Services**

5292 We present background information on IWSs describing how they differ from tra-
5293 ditional web services, the dimensions of their evolution and the currently limited
5294 configuration options available to users.

5295 **11.3.1 ‘Intelligent’ vs ‘Traditional’ Web Services**

5296 Unlike conventional web services, IWSs are built using AI-based components. These
5297 components are unlike traditional software engineering paradigms as they are data-
5298 dependent and do not result in deterministic outcomes. These services make future
5299 predictions on new data based solely against its training dataset; outcomes are
5300 expressed as probabilities that the inference made matches a label(s) within its
5301 training data. Further, these services are often marketed as forever evolving and
5302 ‘improving’. This means that their large training datasets may continuously update
5303 the prediction classifiers making the inferences, resulting both in probabilistic and
5304 non-deterministic outcomes [88, 162]. Critically for software engineers using the
5305 services, these non-deterministic aspects have not been sufficiently documented in
5306 the service’s API documented, which has been shown to confuse developers [91].

5307 A strategy to combat such service changes, which we often observe in traditional
5308 software engineering practices, are for such services to be versioned upon substantial
5309 change. Unfortunately emerging evidence indicates that prominent cloud vendors
5310 providing these IWSs do not release new versioned endpoints of the APIs when the
5311 *internal model* changes [88]. For IWSs, it is impossible to invoke requests specific
5312 to a particular version model that was trained at a particular date in time. This means
5313 that developers need to consider how evolutionary changes to the IWSs they make
5314 use of may impact their solutions *in production*.

5315 **11.3.2 Dimensions of Evolution**

5316 The various key dimensions of the evolution of IWSs is illustrated in Figure 11.2.
5317 There are two primary dimensions of evolution: *changes to the label sets* returned
5318 per image submitted and *changes to the confidences* per label in the set of labels
5319 returned per image. In the former, we identify two key aspects: cardinality changes
5320 and ontology changes. Cardinality changes occur when the service either introduces
5321 or drops a label for the same image at two different generations. Alternatively, the
5322 cardinality may remain stagnant, although this is not guaranteed. This results in
5323 an expectation mismatch by developers as to what labels can or will be returned by
5324 the service. For instance, the terms ‘black’ and ‘black and white’ may be found to
5325 be categorised as two separate labels. Secondly, the ontologies of these labels are

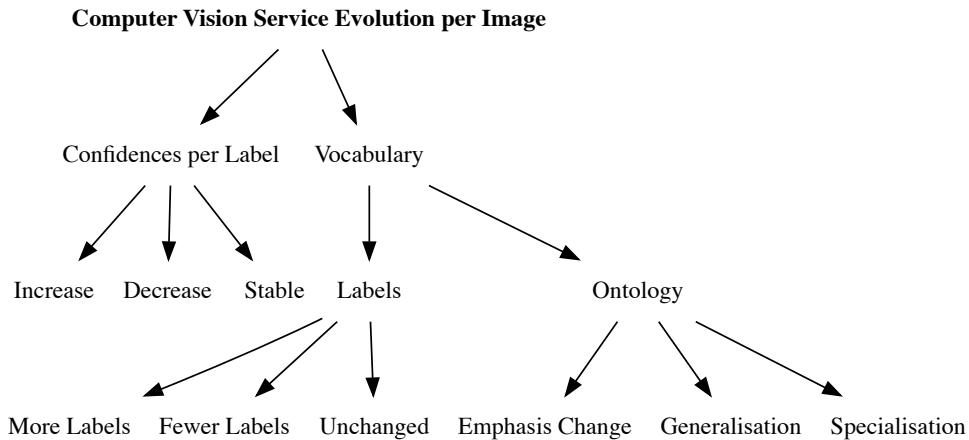


Figure 11.2: The dimensions of evolution identified within CVSSs.



Figure 11.3: A significant confidence increase ($\delta = +0.425$) from ‘window’ (0.559) to ‘water transportation’ (0.984) goes beyond simple decision boundaries.

5326 non-static, and a label may become more generalised into a hypernym, specialised
 5327 into a hyponym, or the emphasis of the label may change either to a co-hyponym or
 5328 another aspect in the image, such as the colour or scene, rather than the subject of
 5329 the image [88].

5330 Secondly, we have identified that the confidence values returned per label are also
 5331 non-static. While some services may present minor changes to labels’ confidences
 5332 resulting from statistical noise, other labels had significant changes that were beyond
 5333 basic decision boundaries. An example is shown in Figure 11.3. Developer code
 5334 written to assume certain ranges/confidence intervals will fail if the service evolves
 5335 in this way.

5336 11.3.3 Limited Configurability

5337 As an example, consider Figure 11.5, which illustrates an image of a dog uploaded to
 5338 a well-known cloud-based CVS. Developers have very few configuration parameters
 5339 in the upload payload (`url` for the image to analyse and `maxResults` for the number
 5340 of objects to detect). The JSON output payload provides the confidence value of its
 5341 estimated bounding box and label of the dog object via its `score` field (0.792). This
 5342 value indicates the level of confidence in the label returned, and is dependent on the
 5343 input to the underlying ML model used by that service. Developers set thresholds

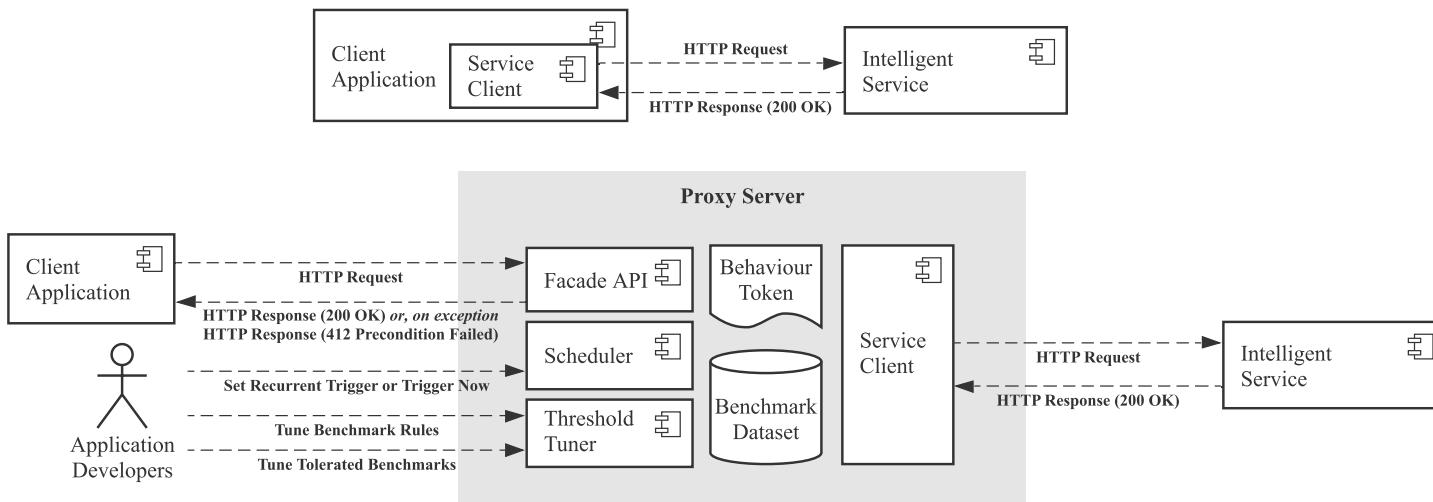


Figure 11.4: Top: Accessing an intelligent service directly. Bottom: Primary components of the Proxy Server approach.

as a decision boundary in this case, a threshold of “greater than 0.7” could indicate that the image contains a dog where as any other value the system is uncertain. These decision boundaries determine if the service’s output is accepted or rejected. However, these confidence scores change whenever a model is re-trained and these changes are not communicated or propagated to developers [88]. Developers can only modify these parameters to influence the score to improve the performance of the IWS. This is unlike many machine learning toolkit hyper-parameter optimisation facilities, which can be used to configure the internal parameters of the algorithm for training a model. In this case, developers using the IWS have no insight into which hyperparameters were used when training the model or the algorithm selected, and cannot tune the trained model. Thus an evaluation procedure must be followed as a part of using an intelligent service for an application to tune their output confidence values. and select appropriate threshold boundaries. While some service providers provide some guidance to thresholding,¹ they do not provide domain-specific tooling. This is because choice of appropriate thresholds is dependent on the data and must consider factors, such as algorithmic performance, financial cost, and impact of false-positives/negatives.

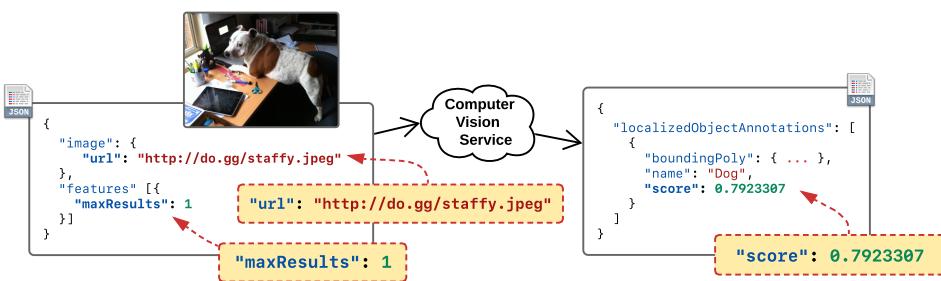


Figure 11.5: Request and response for an intelligent computer vision web service with only three configuration parameters: the image’s url, maxResults and score.

However, decision boundaries in service client code using simple If conditions around confidence scores is not a sufficient enough strategy, as evidence shows intelligent, non-deterministic web services change sporadically and unknowingly. Most traditional, deterministic code bases handle unexpected behaviour of called APIs via *error codes* and exception handling. Thus the non-deterministic components of the client code, such as those using CVSs, will also tend to conflict with their traditional deterministic components as the latter do not deal in terms of probabilities but in using error codes. This makes achieving robust component integration in client code bases hard. More sophisticated monitoring of IWSs in client code is therefore required to map the non-deterministic service behaviour changes to errors such that the surrounding infrastructure can support it and reduce interface boundary problems. While data science literature acknowledges the need for such an architecture [112] they do not offer any technical software engineering solutions to mitigate the issues such that software engineers have a pattern to work against it. To date, there do not

¹<https://bit.ly/36oMgWb> last accessed 20 May 2020.

Table 11.1: Potential reasons for a 412 Precondition Failed response.

Error Code	Error Description
No Key Yet	This indicates that the Proxy Server is still initialising its first behaviour token, i.e., k_0 does not yet exist.
Service Mismatch	The service encoded within the behaviour token provided to the Proxy Server does not match the service the Proxy Server is benchmarked against. This makes it possible for one Proxy Server to face multiple CVSs.
Dataset Mismatch	The benchmark dataset B encoded within the behaviour token does not match the benchmark dataset encoded within the Proxy Server.
Success Mismatch	The success of each response within the benchmark dataset must be true for a behaviour token to be used within a request. This error indicates that k_r is, therefore, not successful.
Min Confidence Mismatch	The minimum confidence delta threshold set in k_t does not match that of k_r .
Max Labels Mismatch	The maximum label delta threshold set in k_t does not match that of k_r .
Response Length Mismatch	The number of responses within k_t does not match that within k_r .
Label Delta Mismatch	An image within B has either dropped or gained a number of labels that exceeds the maximum label delta. Thus, k_r exceeds the threshold encoded within k_t .
Confidence Delta Mismatch	One of the labels within an image encoded in k_r exceeds the confidence threshold encoded within k_t .
Expected Labels Mismatch	One of the expected labels for an image within k_t is now missing.

⁵³⁷⁵ yet exist IWS client code architectures, tactics or patterns that achieve this goal.

⁵³⁷⁶ 11.4 Our Approach

⁵³⁷⁷ To address the requirements from Section 11.2 we have developed a new Proxy
⁵³⁷⁸ Service² that includes: (i) evaluation of an intelligent service using an application
⁵³⁷⁹ specific benchmark dataset, (ii) a Proxy Server to provide client applications with
⁵³⁸⁰ evolution aware errors, and (iii) a scheduled evolution detection mechanism. The
⁵³⁸¹ current approach of using an intelligent API via direct access is shown in Figure 11.4
⁵³⁸² (top). In contrast, an overview of our approach is shown in Figure 11.4 (bottom).
⁵³⁸³ The following sections describe our approach in detail.

⁵³⁸⁴ 11.4.1 Core Components

⁵³⁸⁵ For the purposes of this paper we assume that the intelligent service of interest
⁵³⁸⁶ is an image recognition service, but our approach generalises to other intelligent,

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

5387 trained model-based services e.g., natural language processing, document recogni-
 5388 tion, voice, etc. Each image, when uploaded to the intelligent service returns a
 5389 response (R) which is a set describing a label (l) of what is in the image (i) along
 5390 with its associated confidence (c)—thus $R_i = \{(l_1, c_1), (l_2, c_2), \dots (l_n, c_n)\}$. Most
 5391 documentation of these services imply that these confidence values are all what is
 5392 needed to handle evolution in their systems. This means that if a label changes
 5393 beyond a certain threshold, then the developer can deal with the issue then (or ignore
 5394 it). While this approach may work in some simple application contexts, in many it
 5395 may not. Our Proxy Server offers a way to monitor if these changes go beyond a
 5396 threshold of tolerance, checking against a domain-specific dataset over time.

5397 11.4.1.1 Benchmark Dataset

5398 Monitoring an intelligent service for behaviour change requires a Benchmark Dataset,
 5399 a set of n images. For each image (i) in the Benchmark Dataset (B) there is an associ-
 5400 ated label (l) that represents the true value for that item; $B_i = \{(i_1, l_1), (i_2, l_2), \dots (i_n, l_n)\}$.
 5401 This dataset is used to check for evolution in IWSs by periodically sending each im-
 5402 age within the dataset to the service’s API, as per the rules encoded within the
 5403 Scheduler (see Section 11.4.1.6). By using a dataset specific to the application
 5404 domain, developers can detect when evolution affects their application rather than
 5405 triggering all non-impactful changes. This helps achieve our requirement *R3. Monitor*
5406 the evolution of IWSs for changes that affect the application’s behaviour. Using
 5407 application-specific datasets also ensures that the architectural style can be used for
 5408 different IWSs as only the data used needs to change. This design choice encourages
 5409 reuse satisfying requirement *R4. Implement a flexible architecture that is adapt-*
5410 able to different IWSs and application contexts to facilitate reuse. We propose an
 5411 initial set of guidelines on how to create and update the benchmark dataset within
 5412 Section 11.6.3.1.

5413 11.4.1.2 Facade API

5414 An architectural ‘facade’ is the central component to our mitigation strategy for
 5415 monitoring and detecting for changes in called IWSs. The facade acts as a guarded
 5416 gateway to the intelligent service that defends against two key issues: (i) potential
 5417 shifts in model variations that power the cloud vendor services, and (ii) ensures that
 5418 a context-specific dataset specific to the application being developed is validated
 5419 *over time*. By using a facade we can return evolution-aware error codes to the client
 5420 application satisfying requirement *R1. Define a set of error conditions that specify*
5421 the types of evolution that occur for an intelligent service and enabling requirement
5422 R3. Monitor the evolution of IWSs for changes that affect the application’s behaviour.
 5423 This works by ensuring every request made by the client application contains a valid
 5424 Behaviour Token (see Section 11.4.1.4) and will reject the request when evolution
 5425 has been identified by the Scheduler with an associated error code. The Facade API
 5426 essentially ‘blocks’ the client application out from accessing the intelligent service
 5427 when an invalid state has occurred.

Table 11.2: Rules encoded within a Behaviour Token.

Rule	Description
Max Labels	The value of n .
Min Confidence	The smallest acceptable value of c .
Max δ Labels	The minimum number of labels dropped or introduced from the current k_t and provided k_r to be considered a violation (i.e $ l(k_t) \Delta l(k_r) $).
Max δ Confidence	The minimum confidence change of <i>any</i> label from the current k_t and provided k_r to be considered a violation.
Expected Labels	A set of labels that every response must include.

5428 11.4.1.3 Threshold Tuner

5429 Selecting an appropriate threshold for detecting behavioural change depends on the
 5430 application context. Setting the threshold too low increases the likelihood of incor-
 5431 rect results, while setting the threshold too high means undesired changes are being
 5432 detected. Our approach enables developers to configure these parameters through a
 5433 Threshold Tuner, and consider competing factors such as algorithmic performance,
 5434 financial cost, and impact of false-positives/negatives. This component improves
 5435 robustness as now there is a systematic approach for monitoring and responding to
 5436 incorrect thresholds. Configurable thresholds meet our key requirements *R2* and *R3*.
 5437 An example of the component is detailed within our complement paper published in
 5438 the ESEC/FSE 2020 demonstrations track [89].

5439 11.4.1.4 Behaviour Token

5440 The Behaviour Token stores the current state of the Proxy Server by encoding specific
 5441 rules regarding the evolution of the intelligent service. The current token (at time t)
 5442 held by the Proxy Server is denoted by k_t . These rules are specified by the developer
 5443 upon initialisation of this Proxy Server, and are presented in Table 11.2. When the
 5444 Proxy Server is first initialised (i.e., at $t = 0$), the first Behaviour Token is created
 5445 based on the Benchmark Dataset and its configuration parameters (Table 11.2) and
 5446 is stored locally (thus k_0 is created). The Behaviour Token is passed to the client
 5447 application to be used in subsequent requests to the proxy server, where k_r represents
 5448 the Behaviour Token passed from the client application to the proxy server. Each
 5449 time the proxy server receives the Behaviour Token from the client the validity of the
 5450 token is validated with a comparison to the Proxy Server's current behaviour token
 5451 (i.e., $k_r \equiv k_t$). An invalid token (i.e., when $k_r \not\equiv k_t$) indicates that an error caused by
 5452 evolution has occurred and the application developer needs to appropriately handle
 5453 the exception. Behaviour Tokens are essential for meeting requirement *R3*. *Monitor*
 5454 *the evolution of IWSs for changes that affect the application's behaviour.*

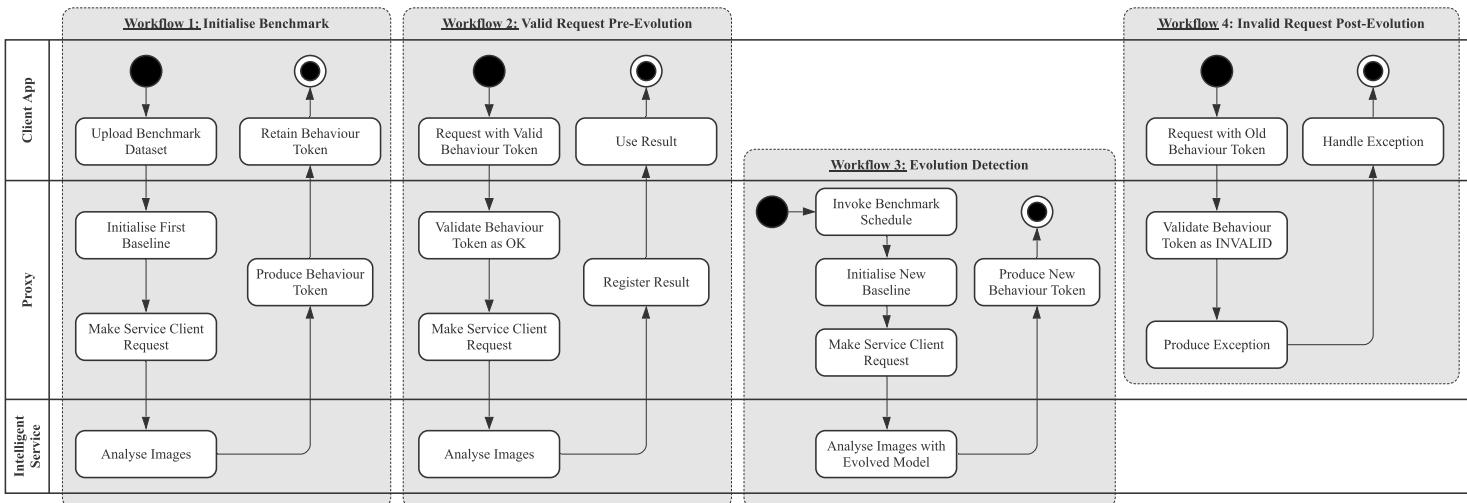


Figure 11.6: State diagram for the four workflows presented.

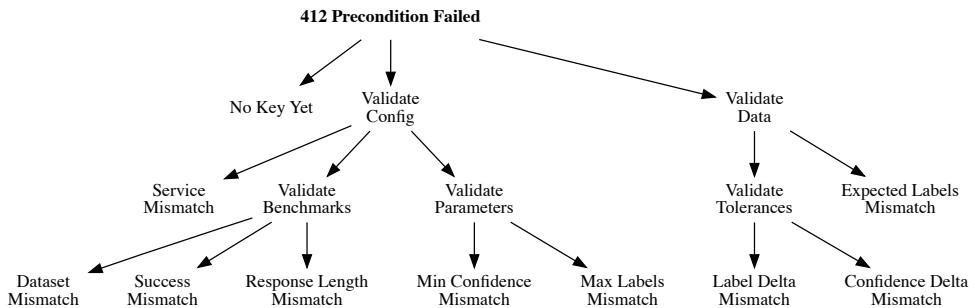


Figure 11.7: Precondition failure taxonomy; leaf nodes indicate error types returned to users.

5455 11.4.1.5 Service Client

5456 If any of the rules above are violated, then the response of the facade request varies
 5457 depending on the behaviour encoded within the behaviour token. This can be one
 5458 of:

- 5459 • **Error:** Where a HTTP non-200 code is returned by the facade to the client
 5460 application, indicating that the client application must deal with the issue
 5461 immediately;
- 5462 • **Warning:** Where a warning ‘callback’ endpoint is called with the violated
 5463 response to be dealt with, but the response is still returned to the client
 5464 application;
- 5465 • **Info:** Where the violated response is logged in the facade’s logger for the
 5466 developer to periodically read and inspect, and the response is returned to the
 5467 client application.

5468 We implement this Proxy Server pattern using HTTP conditional requests. As
 5469 we treat the Label as a first class citizen, we return the labels for a specific image
 5470 (r_i) only where the *Entity Tag* (ETag) or *Last Modified* validators pass. The k_r
 5471 is encoded within either the ETag (i.e., a unique identifier representing t) or as
 5472 the date labels (and thus models) were last modified (i.e., using the *If-Match*
 5473 or *If-Unmodified-Since* conditional headers). We note that the use of *weak*
 5474 ETags should be used, as byte-for-byte equivalence is not checked but only semantic
 5475 equivalence within the tolerances specified. Should t evolve to an invalid state (i.e.,
 5476 k_r is no longer valid against k_t) then the behaviour as described above will be
 5477 enacted.

5478 These HTTP header fields are used as the ‘backbone’ to help enforce robustness
 5479 of the services against evolutionary changes and context within the problem domain
 5480 dataset. Responses from the service are forwarded to the clients when such rules
 5481 are met, otherwise alternative behaviour occurs. For example, the most severe of
 5482 violated erroneous behaviour is the ‘Error’ behaviour. To enforce this rule, we
 5483 advocate for use of the 412 Precondition Failed HTTP error if a violation
 5484 occurs, as a *If-** conditional header was violated. An example of this architectural
 5485 pattern with the ‘Error’ behaviour is illustrated in Figure 11.6.

5486 We suggest the 412 Precondition Failed HTTP error be returned in the
5487 event that a behaviour token is violated against a new benchmark. Further details
5488 outlining the reasons why a precondition has failed are encoded within a JSON
5489 response sent back to the consuming application. The following describes the
5490 two broad categories of possible errors returned: *robustness precondition failure*
5491 or *benchmark precondition failure*. These are illustrated in a high level within
5492 Figure 11.7 where leaf nodes are the potential error types that can be returned. A
5493 list of the different error codes are given in Table 11.1, where errors above the rule
5494 are robustness expectations (which check for basic requirements such as whether the
5495 key provided encodes the same data as the dataset in the facade) while those below
5496 are benchmark expectations (which identifies evolution cases).

5497 11.4.1.6 Scheduler

5498 The Scheduler is responsible for triggering the Evolution Detection Workflow (de-
5499 scribed in detail below in Section 11.4.2). Developers set the schedule to run in
5500 the background at regular intervals or to trigger if violations occur z times. The
5501 Scheduler is the component that enables our architectural style to identify called
5502 intelligent service software evolution and to notify the client applications that such
5503 evolution has occurred. Client applications can then respond to this evolution in a
5504 timely manner rather than wait for the system to fail, as in our motivating example.
5505 The Scheduler is necessary to satisfy our requirements $R2$ and $R3$.

5506 11.4.2 Usage Example

5507 We explain how developer Michelina, from our motivating example, would use
5508 our proposed solution to satisfy the requirements described in Section 11.2. Each
5509 workflow is presented in Figure 11.6. Only *Workflow 1 - Initialise Benchmark* is
5510 executed once, while the rest are cycled. The description below assumes Michelina
5511 has implemented the Proxy.

5512 11.4.2.1 Workflow 1. Initialise Benchmark

5513 The first task that Michelina has to do is to prepare and initialise the benchmark
5514 dataset within the Proxy Server. To prepare a representative dataset, Michelina needs
5515 to follow well established guidelines such as those proposed by Pyle. Michelina also
5516 needs to manually assign labels to each image before uploading the dataset to the
5517 Proxy along with the thresholds to use for detecting behavioural change. The full set
5518 of parameters that Michelina has to set are based on the rules shown in Table 11.2.
5519 Michelina cannot use the Proxy to notify her of evolution until a Benchmark Dataset
5520 has been provided. The Proxy then sends each image in the Benchmark Dataset to
5521 the intelligent service and stores the results. From these results, a Behaviour Token
5522 is generated which is passed back to the Client Application. Michelina uses this
5523 token in all future requests to the Proxy as the token captures the current state of the
5524 intelligent service.

5525 *11.4.2.2 Workflow 2. Valid Request Pre-Evolution*

5526 Workflow 2 represents the steps followed when the intelligent service is behaving as
5527 expected. Michelina makes a request to label an image to the Proxy using the token
5528 that she received when registering the Benchmark Dataset. The token is validated
5529 with the Proxy’s current state token and then a request to label the image is made to
5530 the intelligent service if no errors have occurred. Results returned by the intelligent
5531 service are registered with the Proxy Server. Michelina can be confident that the
5532 result returned by our service is in line with her expectations.

5533 *11.4.2.3 Workflow 3. Evolution Detection*

5534 Workflow 3 describes how the Proxy functions when behavioural change is present
5535 in the called intelligent service. Michelina sets a schedule for once a day so that the
5536 Proxy’s Scheduler triggers Workflow 3. First, each image in the Benchmark Dataset
5537 is sent to the intelligent service. Unlike, Workflow 1, we already have a Behaviour
5538 Token that represents the previous state of the intelligent service. In this case, the
5539 model behind the intelligent service has been updated and provides different results
5540 for the Benchmark Dataset. Second, the Proxy updates the internal Behaviour Token
5541 ready for the next request. At this stage Michelina will be notified that the behaviour
5542 of the intelligent service has changed.

5543 *11.4.2.4 Workflow 4. Invalid Request Post-Evolution*

5544 Workflow 4 provides Michelina with an error message when evolution has been
5545 detected. Michelina’s client application makes a request to the Proxy Server with
5546 an old Behaviour Token. The Proxy Server then validates the client token which is
5547 invalid as the Behaviour Token has been updated. In this case, an exception is raised
5548 and an appropriate error message as discussed above is included in the response
5549 back to Michelina’s client application. Michelina can code her application to handle
5550 each error class in appropriate ways for her domain.

5551 **11.5 Evaluation**

5552 Our evaluation of our novel intelligent service Proxy Server approach uses a technical
5553 evaluation based on the results of an observational study. We used existing datasets
5554 from observational studies [88, 217] to identify problematic evolution in computer
5555 vision labelling services. This technical evaluation is designed to show: (i) what
5556 the responses are with and without our architecture present (highlighting errors); (ii)
5557 the overall increased robustness using enhanced responses; and (iii) the technical
5558 soundness of the approach. Thus, we propose the following research question which
5559 we answer in Section 11.5.2: “*Can the architecture identify evolutionary issues of*
5560 *computer vision services via error codes?*” Based on our findings we proposed and
5561 implemented the Proxy Server using a Ruby development framework which we have

5562 made available online for experimentation.³ Additional data was collected from the
5563 CVS and sent to the Proxy Server to evaluate how the service handles behavioural
5564 change.

5565 11.5.1 Data Collection and Preparation

5566 To minimise reviewer bias, we do not identify the name of the service used, however
5567 this service was one of the most adopted cloud vendors used in enterprise applications
5568 in 2018 [119]. The two existing datasets used [88, 217] consisted of 6,680 images.

5569 We initialised the benchmark (workflow 1) in November 2018, and sent each
5570 image to the service every eight days and captured the JSON responses through the
5571 facade API (workflow 2) until March 2019. This resulted in 146,960 JSON responses
5572 from the target CVS. We then selected the first and last set of JSON responses (i.e.,
5573 13,360 responses) and independently identified 331 cases of evolution of the original
5574 6,680 images. This was achieved by analysing the JSON responses for each image
5575 taken in using an evaluation script.⁴

5576 For each JSON response, evolution (as classified by Figure 11.2) was determined
5577 either by a vocabulary or confidence per label change in the first and last responses
5578 sent. For the 331 evolving responses, we calculated the delta of the label's confidence
5579 between the two timestamps and the delta in the number of labels recorded in the
5580 entire response. Further, for the highest-ranking label (by confidence), we manually
5581 classified whether its ontology became more specific, more generalised or whether
5582 there was substantial emphasis change. The distribution of confidence differences per
5583 these three groups are shown in Figure 11.8, with the mean confidence delta indicated
5584 with a vertical dotted line. This highlights that, on average, labels that change
5585 emphasis generally have a greater variation, such as the example in Figure 11.3.
5586 Further, we grouped each image into one of four broad categories—*food*, *animals*,
5587 *vehicles*, *humans*—and assessed the breakdown of ontology variance as provided
5588 in Table 11.3. We provide this dataset as an additional contribution and to permit
5589 replication.⁵ The parameters set for our initial benchmark were a delta label value of
5590 3 and delta confidence value of 0.01. Expected labels for relevant groups were also
5591 assigned as mandatory label sets (e.g., *animal* images used ‘animal’, ‘fauna’ and
5592 ‘organism’; *human* images used ‘human’ etc.).

5593 11.5.2 Results

5594 Examples of the March 2019 responses contrasting the proxy and direct service
5595 responses in our evaluation are shown in Figures 11.9 to 11.11. (Due to space limita-
5596 tions, the entire JSON response is partially redacted using ellipses.) These examples
5597 identify the label identified with the highest level of confidence in three examples
5598 against the ground truth label in the benchmark dataset. In total, the Proxy Server
5599 identified 1,334 labels added to the responses and 1,127 labels dropped, with, on
5600 average, a delta of 8 labels added. The topmost labels added were ‘architecture’

³<http://bit.ly/2TIMmDh> last accessed 5 March 2020.

⁴<http://bit.ly/2G7saFJ> last accessed 2 March 2020.

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

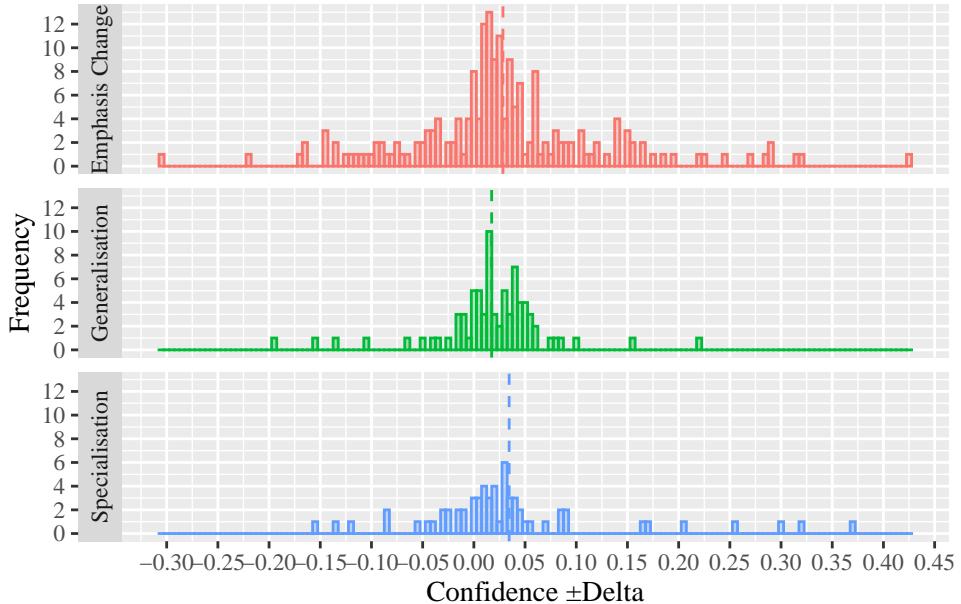


Figure 11.8: Histogram of confidence variation.

5601 at 32 cases, ‘building’ at 20 cases and ‘ingredient’ at 20 cases; the topmost
 5602 labels dropped were ‘tree’ at 21 cases, ‘sky’ at 19 cases and ‘fun’ at 17 cases.
 5603 1054 confidence changes were also observed by the Proxy Server, on average a delta
 5604 increase of 0.0977.

5605 In Figure 11.9, we highlight an image of a sheep that was identified as a ‘sheep’
 5606 (at 0.9622) in November 2018 and then a ‘mammal’ in March 2019. This evolution
 5607 was classified by the Proxy Server as a confidence change error as the delta in
 5608 the confidences between the two timestamps exceeds the parameter set of 0.01—in
 5609 this case, ‘sheep’ was downgraded to the third-ranked label at 0.9816, thereby
 5610 increasing by a value of 0.0194. As shown in the example, four other labels evolved
 5611 for this image between the two time stamps (‘herd’, ‘livestock’, ‘terrestrial
 5612 animal’ and ‘snout’) with an average increase of 0.1174 found. Such information
 5613 is encoded as a 412 HTTP error returned back to the user by the Proxy Server,
 5614 rejecting the request as substantial evolution has occurred, however the response
 5615 directly from the service indicates no error at all (indicating by a 200 HTTP response).

5616 Similarly, Figure 11.10 shows a violation of the number of acceptable changes in
 5617 the number of labels a response should have between two timestamps. In November
 5618 2018, the response includes the labels ‘car’, ‘motor vehicle’, ‘city’ and
 5619 ‘road’, however these labels are not present in the 2019 response. The response
 5620 in 2019 introduces ‘transport’, ‘building’, ‘architecture’, and ‘house’.
 5621 Therefore, the combined delta is 4 dropped and 4 introduced labels, exceeding our
 5622 threshold set of 3.

5623 Lastly, Figure 11.11 indicates an expected label failure. In this example, the
 5624 label ‘fauna’ was dropped in the 2018 label set, which was an expected label
 5625 of all animals we labelled in our dataset. Additionally, this particular response

Table 11.3: Variance in ontologies for the five broad categories.

Ontology Change	Food	Animal	Vehicles	Humans	Other	Total
Generalisation	8	13	11	8	38	78
Specialisation	5	12	1	1	43	62
Emphasis Change	18	4	10	21	138	191
Total	31	29	22	30	219	331

5626 introduced ‘green iguana’, ‘iguanidae’, and ‘marine iguana’ to its label
5627 set. Therefore, not only was this response in violation of the label delta mismatch, it
5628 was also in violation of the expected labels mismatch error, and thus is caught twice
5629 by the Proxy Server.

5630 11.5.3 Threats to Validity

5631 11.5.3.1 Internal Validity

5632 As mentioned, we selected a popular CVS provider to test our proxy server against.
5633 However, there exist many other CVSs, and due to language barriers of the authors,
5634 no non-English speaking service were selected despite a large number available from
5635 Asia. Further, no user evaluation has been performed on the architectural tactic so
5636 far, and therefore developers may suggest improvements to the approach we have
5637 taken in designing our tactic. We intend to follow this up with a future study.

5638 11.5.3.2 External Validity

5639 This paper only evaluates the object detection endpoint of a computer vision-based
5640 intelligent service. While this type of intelligent service is one of the more mature
5641 AI-based services available on the market—and is largely popular with develop-
5642 ers [91]—further evaluations of the our tactic may need to be explored against other
5643 endpoints (i.e., object localisation) or, indeed, other types of services, such as natural
5644 language processing, audio transcription, or on time-series data. Future studies may
5645 need to explore this avenue of research.

5646 11.5.3.3 Construct Validity

5647 The evaluation of our experiment was largely conducted under clinical conditions,
5648 and a real-world case study of the design and implementation of our proposed tactic
5649 would be beneficial to learn about possible side-effects from implementing such a
5650 design (e.g., implications to cost etc.). Therefore, our evaluation does not consider
5651 more practical considerations that a real-world, production-grade system may need
5652 to consider.



Label: Animal
Nov 2018: 'sheep' (0.9622)
Mar 2019: 'mammal' (0.9890)
Category: Confidence Change

Intelligent Service Response in March 2019

```

1 { "responses": [ { "label_annotations": [
2   { "mid": "/m/04rky",
3     "description": "mammal",
4     "score": 0.9890478253364563,
5     "topicality": 0.9890478253364563 },
6   { "mid": "/m/09686",
7     "description": "vertebrate",
8     "score": 0.9851104021072388,
9     "topicality": 0.9851104021072388 },
10  { "mid": "/m/07bgp",
11    "description": "sheep",
12    "score": 0.9815810322761536,
13    "topicality": 0.9815810322761536 },
14    ... ] } ]

```

Proxy Server Response in March 2019

```

1 { "error_code": 8,
2   "error_type": "CONFIDENCE_DELTA_MISMATCH",
3   "error_data": {
4     "source_key": { ... },
5     "source_response": { ... },
6     "violating_key": { ... },
7     "violating_response": { ... },
8     "delta_confidence_threshold": 0.01,
9     "delta_confidences_detected": {
10       "sheep": 0.01936030388219212,
11       "herd": 0.15035879611968994,
12       "livestock": 0.13112884759902954,
13       "terrestrial_animal": 0.1791478991508484,
14       "snout": 0.10682523250579834
15     },
16     "uri": "http://localhost:4567/demo/data/000000005992.jpeg"
17     ↵ ,
      "reason": "Exceeded confidence delta threshold ±0.01 in 5
      ↵ labels (delta mean=+0.1174)." } }

```

Figure 11.9: Example of substantial confidence change due to evolution.



Label: Vehicle
Nov 2018: 'vehicle' (0.9045)
Mar 2019: 'motorcycle' (0.9534)
Category: Label Set Change

Intelligent Service Response in March 2019

```

1 { "responses": [ { "label_annotations": [
2   { "mid": "/m/07yv9",
3     "description": "vehicle",
4     "score": 0.9045347571372986,
5     "topicality": 0.9045347571372986 },
6   { "mid": "/m/07bsy",
7     "description": "transport",
8     "score": 0.9012271165847778,
9     "topicality": 0.9012271165847778 },
10  { "mid": "/m/0dx1j",
11    "description": "town",
12    "score": 0.8946694135665894,
13    "topicality": 0.8946694135665894 },
14    ... ] } ] }
```

Proxy Server Response in March 2019

```

1 { "error_code": 7,
2   "error_type": "LABEL_DELTA_MISMATCH",
3   "error_data": {
4     "source_key": { ... },
5     "source_response": { ... },
6     "violating_key": { ... },
7     "violating_response": { ... },
8     "delta_labels_threshold": 5,
9     "delta_labels_detected": 8,
10    "uri": "http://localhost:4567/demo/data/000000019109",
11    "new_labels": [ "transport", "building", "architecture", "
12      ↪ house" ],
12    "dropped_labels": [ "car", "motor vehicle", "city", "road"
13      ↪ ],
13    "reason": "Exceeded label count delta threshold ±5 (4 new
13      ↪ labels + 4 dropped labels = 8)." } }
```

Figure 11.10: Example of substantial changes of a response's label set due to evolution.



Label: Fauna
Nov 2018: 'reptile' (0.9505)
Mar 2019: 'iguania' (0.9836)
Category: Ontology Specialisation

Intelligent Service Response in March 2019

```

1 | { "responses": [ { "label_annotations": [
2 |   { "mid": "/m/08_jw6",
3 |     "description": "iguania",
4 |     "score": 0.9835183024406433,
5 |     "topicality": 0.9835183024406433 },
6 |   { "mid": "/m/06bt6",
7 |     "description": "reptile",
8 |     "score": 0.9833670854568481,
9 |     "topicality": 0.9833670854568481 },
10 |   { "mid": "/m/01vq7_",
11 |     "description": "iguana",
12 |     "score": 0.9796721339225769,
13 |     "topicality": 0.9796721339225769 },
14 |   ... ] } ]

```

Proxy Server Response in March 2019

```

1 | { "error_code": 9,
2 |   "error_type": "EXPECTED_LABELS_MISMATCH",
3 |   "error_data": {
4 |     "source_key": { ... },
5 |     "violating_response": { ... },
6 |     "uri": "http://localhost:4567/demo/data/0052",
7 |     "expected_labels": [ "fauna" ],
8 |     "labels_detected": [ "iguana", "green iguana", "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 11.11: Example of an expected label missing due to evolution.

5653 11.6 Discussion**5654 11.6.1 Implications****5655 11.6.1.1 For cloud vendors**

5656 Cloud vendors that provide IWSs may wish to adopt the architectural tactic presented
5657 in this paper by providing a proxy, auxiliary service (or similar) to their existing ser-
5658 vices, thereby improving the current robustness of these services. Further, they
5659 should consider enabling developers of this technical domain knowledge by pre-
5660 venting client applications from using the service without providing a benchmark
5661 dataset, such that the service will return HTTP error codes. These procedures should
5662 be well-documented within the service’s API documentation, thereby indicating to
5663 developers how they can build more robust applications with their IWSs. Lastly,
5664 cloud vendors should consider updating the internal machine learning models less
5665 frequently unless substantial improvements are being made. Many different appli-
5666 cations from many different domains are using these IWSs so it is unlikely that
5667 the model changes are improving all applications. Versioned endpoints would help
5668 with this issue, although—as we have discussed—context using benchmark datasets
5669 should be provided.

5670 11.6.1.2 For application developers

5671 Developers need to monitor all IWSs for evolution using a benchmark dataset and
5672 application specific thresholds before diving straight into using them. It is clear that
5673 the evolutionary issues have significant impact in their client applications [88], and
5674 therefore they need to check the extent this evolution has between versions of an
5675 intelligent service (should versioned APIs be available). Lastly, application devel-
5676 opers should leverage the concept of a proxy server (or other form of intermediary)
5677 when using IWSs to make their applications more robust.

5678 11.6.1.3 For project managers

5679 Project managers need to consider the cost of evolution changes on their application
5680 when using IWSs, and therefore should schedule tasks for building maintenance
5681 infrastructure to detect evolution. Consider scheduling tasks that evaluates and
5682 identifies the frequency of evolution for the specific intelligent service being used.
5683 Our research we have found some IWSs that are not versioned but rarely show
5684 behavioural changes due to evolution.

5685 11.6.2 Limitations

5686 In the situation where a solo developer implements the Proxy Service the main
5687 limitation is the cost vs response time trade-off. Developers may want to be notified
5688 as soon as possible when a behavioural change occurs which requires frequent
5689 validation of the Benchmark Dataset. Each time the Benchmark Dataset is validated
5690 each item is sent as a request to the intelligent service. As cloud vendors charge

5691 per request to an intelligent service there are financial implications for operating
5692 the Proxy Service. If the developer optimises for cost then the application will take
5693 longer to respond to the behavioural change potentially impact end users. Developers
5694 need to consider the impact of cost vs response time when using the Proxy Service.

5695 Another limitation of our approach is the development effort required to imple-
5696 ment the Proxy Service. Developers need to build a scheduling component, batch
5697 processing pipeline for the Benchmark Dataset, and a web service. These com-
5698 ponents require developing and testing which impact project schedules and have
5699 maintenance implications. Thus, we advise developers to consider the overhead of
5700 a Proxy Service and way up the benefits with have incorrect behaviour caused by
5701 evolution of IWSs.

5702 **11.6.3 Future Work**

5703 *11.6.3.1 Guidelines to construct and update the Benchmark Dataset*

5704 Our approach assumes that each category of evolution is present in the Benchmark
5705 Dataset prepared by the developer. Further guidelines are required to ensure that the
5706 developer knows how to validate the data before using the Proxy Service. While the
5707 focus of this paper was to present and validate our architectural tactic, guidelines
5708 on how to construct and update benchmark datasets for this tactic will need to be
5709 considered in future work. Data science literature extensively covers dataset prepa-
5710 ration (e.g., [202, 289]), and many example benchmark datasets are readily available
5711 [24, 146, 379]. An initial set of guidelines are proposed as follows: data must be
5712 contextualised and appropriately sampled to be representative of the client applica-
5713 tion in particular the patterns present in the data, contain both positive and negative
5714 examples (this is/is not a cat); where to source data from (existing datasets, Google
5715 Images/Flickr, crowdsourced etc.); whether the dataset is synthetically generated to
5716 increase sample size; and how large a benchmark dataset size should be (i.e., larger
5717 the better but takes more effort and costs more). Benchmark datasets can also be
5718 used by software engineers provided the domain and context is appropriate for their
5719 specific application’s context. Software engineers also benefit from our approach
5720 even if these guidelines are not strictly adhered to provided they use an application-
5721 specific dataset (i.e., data collected from the input source for their application). The
5722 main reason for this is that without our proposed tactic there are limited ways to
5723 build robust software with intelligent services. Future testing and evaluation of these
5724 guidelines should be considered.

5725 *11.6.3.2 Extend the evolution categories to support other IWSs*

5726 This paper has used computer vision services to assess our proposed tactic, and
5727 therefore further investigation is needed into the evolution characteristics of other
5728 IWSs. The evolution challenges with services that provide optimisation algorithms
5729 such as route planning are likely to differ from CVSs. These characteristics of an
5730 application domain have shown to greatly influence software architecture [25] and
5731 further development of the Proxy Service will need to account for these differences.

5732 As an example, we have identified many similar issues that exist for natural language
5733 processing, where topic modelling produces labels on large bodies of text with
5734 associated confidences. Therefore, the *broader* concepts of our contribution (e.g.,
5735 behaviour token parameters, error codes etc.) can be used to handle issues in natural
5736 language processing to demonstrate the generalisability of the architecture to other
5737 intelligent services. We plan to apply our tactic to natural language processing and
5738 other intelligent services in our future work.

5739 *11.6.3.3 Provide tool support for optimising parameters for an application context*

5740 Appropriately using the Proxy Service requires careful selection of thresholds,
5741 benchmark rules and schedule. Further work is required to support the developer in
5742 making these decisions so an optimal application specific outcome is achieved. One
5743 approach is to present the trade-offs to the developer and let them visualise the
5744 impact of their decisions. We have developed an early prototype for such purpose
5745 in [89].

5746 *11.6.3.4 Improvements for a more rigorous approach*

5747 Conducting a more formal evaluation of our proposed architecture would benefit
5748 the robustness of the solution presented. This could be done in various ways,
5749 for example, using a formal architecture evaluation method such as ATAM [191]
5750 or a similar variant [51]; conducting user evaluation via brainstorming sessions or
5751 interviews with practitioners who may provide suggestions to improve our approach;
5752 determining better strategies to fully-automate the approach and reduce manual steps;
5753 and using real-world industry case studies to identify other factors such as cost and
5754 maintenance issues. All these are various avenues of research that would ultimately
5755 benefit in a more well-rounded approach to the architectural tactic we have proposed.

5756 11.7 Related Work

5757 11.7.0.1 Robustness of Intelligent Services

5758 While usage of IWSs have been proven to have widespread benefits to the commu-
5759 nity [94, 298], they are still largely understudied in software engineering literature,
5760 particularly around their robustness in production-grade systems. As an example,
5761 advancements in computer vision (largely due to the resurgence of convolutional
5762 neural networks in the late 1990s [211]) have been made available through IWSs and
5763 are given marketed promises from prominent cloud vendors, e.g., “with Amazon
5764 Rekognition, you don’t have to build, maintain or upgrade deep learning pipelines”.⁶
5765 However, while vendors claim this, the state of the art of *computer vision itself*
5766 is still susceptible to many robustness flaws, as highlighted by many recent stud-
5767 ies [113, 310, 366]. Further, each service has vastly different (and incompatible)
5768 ontologies which are non-static and evolve [88, 265], certain services can mislabel

⁶<https://aws.amazon.com/rekognition/faqs/>, accessed 21 November 2019.

5769 images when as little as 10% noise is introduced [162], and developers have a shallow
5770 understanding of the fundamental AI concepts behind these issues, which presents a
5771 dichotomy of their understanding of the technical domain when contrasted to more
5772 conventional domains such as mobile application development [91].

5773 *11.7.0.2 Proxy Servers as Fault Detectors*

5774 Fault detection is an availability tactic that encompasses robustness of software [31].
5775 Our architecture implements the sanity check and condition monitoring techniques
5776 to detect faults [31, 168], by validating the reasonableness of the response from the
5777 intelligent service against the conditions set out in the rules encoded in the benchmark
5778 dataset and behaviour token. As we do in this study, the proxy server pattern can be
5779 used to both detect and action faults in another service as an intermediary between a
5780 client and a server. For example, addressing accessibility issues using proxy servers
5781 has been widely addressed [42, 43, 349, 390] and, more recently, they have been
5782 used to address in-browser JavaScript errors [108].

5783 **11.8 Conclusions**

5784 IWSs are gaining traction in the developer community, and this is shown with
5785 an evermore growing adoption of CVSs in applications. These services make
5786 integration of AI-based components far more accessible to developers via simple
5787 RESTful APIs that developers are familiar with, and offer forever-‘improving’ object
5788 localisation and detection models at little cost or effort to developers. However, these
5789 services are dependent on their training datasets and do not return consistent and
5790 deterministic results. To enable robust composition, developers must deal with the
5791 evolving training datasets behind these components and consider how these non-
5792 deterministic components impact their deterministic systems.

5793 This paper proposes an integration architectural tactic to deal with these issues
5794 by mapping the evolving and probabilistic nature of these services to deterministic
5795 error codes. We propose a new set of error codes that deal directly with the erroneous
5796 conditions that has been observed in IWSs, such as computer vision. We provide
5797 a reference architecture via a proxy server that returns these errors when they are
5798 identified, and evaluate our architecture, demonstrating its efficacy for supporting
5799 IWS evolution. Further, we provide a labelled dataset of the evolutionary patterns
5800 identified, which was used to evaluate our architecture.

5801

Part III

5802

Postface

CHAPTER 12

5803

5804

5805

Conclusions & Future Work

5806

5807 In this chapter, we provide a summary of the contributions within the body of
5808 this work. We evaluate the significance of the research outcomes to the software
5809 engineering research community and identify potential criticisms of these outcomes.
5810 Lastly, we indicate future avenues of research resulting from this thesis and provide
5811 concluding remarks.

5812 12.1 Contributions of this Work

5813 This thesis has presented three primary contributions to the body of software engi-
5814 neering knowledge. Namely, we have presented an improved understanding in the
5815 landscape of IWSs—concretely, those that provide computer vision—by examining
5816 their runtime behaviour and evolution profile over a longitudinal study (Chapter 4).
5817 The implications of this work emphasise the caution developers need to take be-
5818 fore diving deep into using these services, and highlight the substantial impacts to
5819 software quality if these considerations are ignored. We showed that developers
5820 find working with this software more frustrating when contrasted to conventional
5821 software engineering domains (Chapter 6), and that the distribution of the types of
5822 issues they face differs from that of the types of issues developers face in established
5823 areas such as mobile and web development (Chapter 5). Furthermore, developers
5824 find the completeness of the existing CVS API documentation poor (Chapter 5),
5825 and therefore an investigation into the attributes of what *constitutes* a complete ap-
5826 plication programming interface (API) document according to literature and how
5827 developers respond to the efficacy of these attributes produced a taxonomy that,
5828 when applied to three CVS service providers, found 12 areas of improvement of
5829 the services’ documentation (Chapter 8). This taxonomy further serves as a go-to
5830 ‘checklist’ for any software engineer to review a prioritised list of documentation
5831 elements worth implementing into their own API documentation. Lastly our inves-
5832 tigations into improved intelligent service integration architectures proposes several

strategies by which developers can guard against the non-deterministic evolutionary issues found in Chapter 4. Preliminary solutions such as that presented in Chapter 9 helped informed further investigations into how developers can use a novel workflow to better select appropriate confidence thresholds calibrated for their application’s domain (Chapter 10) and prevent evolution evident in CVSs via a client-server intermediary proxy server strategy (Chapter 11). A more extensive discussion into the contributions of this thesis is presented in Section 1.7.

12.1.1 Answers to Research Questions

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

12.1.1.1 RQ1: “What is the nature of cloud-based CVSs?”

 These services are in nascent stage, are difficult to evaluate, and are not easily interchangeable. They present themselves as conceptually similar, but we find they functionally differ between vendors. Their labels are semantically disparate and work needs to be done on consolidating a standardised vocabulary for labels. Evolution within these services occurs and is not sufficiently versioned or documented to developers as results from services are non-static.

Irrespective of which service is used, the vocabulary used to label an image is disparate. We find that **there exists no common standard vocabulary** (e.g., ‘border collie’ vs. ‘collie’) and **semantic consistency for the same image between services is disparate**, for example as that shown in Figure 12.1 (left). The runtime behaviour of these services when contrasted against *each other* is, therefore, inconsistent, and thus (without semantic comparison of images, such as that suggested in Chapter 9) the vendors are not ‘plug-and-play’. In contrast to deterministic web services, the same result is functionally guaranteed despite which service is used. For instance, conceptually, a cloud storage service will provide the same output for the same input; that is, regardless of whether a developer uses AWS or Google Cloud object storage, when they upload a file, that file is (more or less) guaranteed to be stored. A deterministic input/output is, thereby, conceptually and functionally guaranteed. However, we find that the nature of intelligent services are conceptually similar but functionally different between services, and therefore developers are likely to become vendor locked. For instance, as we show in Section 4.5.1, one service may return the duplicity of objects in an image (e.g., ‘several’), while another service may return the subject of the image (e.g., ‘carrot’) or a hypernym of that subject (e.g., ‘food’), and another service may focus on the environment of the image (e.g., ‘indoors’). Further, even when a label is consistent between services, we find the consistency of how well they agree to that result—as measured by their confidence score in the label—does not always strictly match in their level of agreement. As

5865 we show in Figure 12.1 (right), **distributions of agreement can be disparate even**
5866 **where services agree on a label for the same image.** Lastly, while intelligent
5867 services that provide computer vision are somewhat stable in the responses they
5868 return, **their responses are non-static.** There is no guarantee that a request with
5869 the same image sent in testing will return the same response, and we find that this
5870 potential evolution risk is not sufficiently communicated to developers.

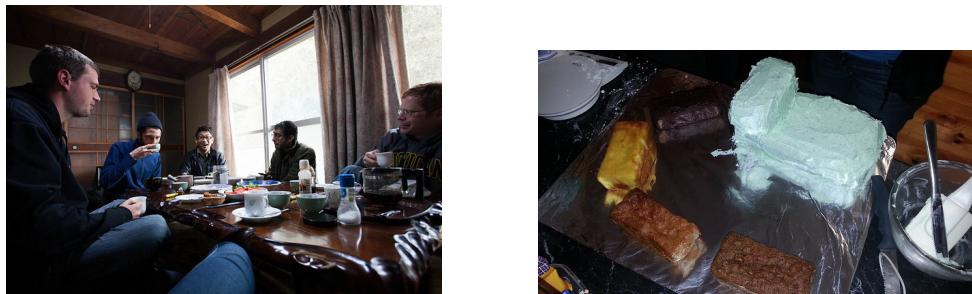


Figure 12.1: *Left:* Semantic consistency between services is not always guaranteed. Two services identified this image as ‘people’, while another identified ‘conversation’, which is not registered at all as a possible label from the other two services. *Right:* Even when services agree on a label, the distribution of their level of agreement is (at times) inconsistent—in the above image, ‘food’ is detected at confidence levels of three services ranging from 94.93% to 41.39%.

5871 12.1.1.2 *RQ2: “Are CVS APIs sufficiently documented?”*

☞ *These services are largely well-documented, but areas of improvement can be identified. By applying the five-dimensional taxonomy we propose in Chapter 8 to three services, we found there to be twelve ways vendors can better improve their services’ documentation. We found the ways in which developers can use these services for their purposes could be improved—such as improved tutorials that integrate multiple components of the service—and by providing better descriptions to improve developers’ conceptual understanding of computer vision.*

5872 To understand if these services are sufficiently documented, we first investigated
5873 what documentation artefacts constitute a complete API document, investigating
5874 literature and validating this against developers using a survey. These consist of five
5875 dimensions: usage description (or *how* developers can use the API); design rationale
5876 (or *when* the developers should use it); domain concepts (or *why* developers should
5877 use it in their application domain); support artefacts (or *what* additional documen-
5878 tation could be provided to support developers); and, documentation presentation
5879 (or *visualisation* of the prior four dimensions). This taxonomy is presented with
5880 further detail in Chapter C. **Developers argue that code snippets are the most**
5881 **important documentation artefact, followed closely by tutorials and low-level**

reference documentation. This is largely covered by existing research. When we apply this taxonomy to intelligent services such as CVSSs, we find that there can be improvements made to all dimensions except documentation presentation, which is sufficient. The largest suggested improvements fall into the usage description dimension, in which quick-start guides, step-by-step tutorials, reference applications, best-practices, listings of all API components, minimum system dependencies, and installation instructions require further detail. The second largest dimension falls into the domain concepts behind computer vision, where vendors should provide a greater emphasis behind computer vision concepts and definitions of relevant computer vision terminology (especially since many vendors refer to the same concept under different terms, such as ‘image tagging’ and ‘label detection’ for what is essentially object recognition). The lack of complete documentation in domain concepts was further reflected in developer discussions on Stack Overflow, as found in Chapter 5. Section 8.6.3 details these suggested improvements in greater detail.

12.1.1.3 RQ3: “Are CVSS more misunderstood than conventional software engineering domains?”

↳ In conventional software engineering domains, where the technical domain is well-established and well-understood by developers, questions asked by developers are of greater depth. In contrast, their shallow understanding of the technical domain of computer vision is reflected by questions that highlight a poor understanding of the behaviour of these services and the contexts by which they work. Thus, simpler questions are asked, such as help with trying to understand basic error codes, or clarification of basic concepts and terminologies in computer vision. Therefore, we argue that they are more misunderstood seeing as the domain of intelligent services is still immature.

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

5913 this is rooted in a need for developers to have some basic understanding of computer
 5914 vision before diving into services such as these. In terms of the documentation of
 5915 these services, **developers express frustration towards the completeness of the**
 5916 **documentation**, whereby they seek additional information from the official docu-
 5917 mentation sources but are unable to find anything to help resolve this gap. Further,
 5918 **they question the accuracy of the cloud documentation since it is in contrast**
 5919 **with the behaviour they observe**, as related to the discrepancy-based issues they
 5920 find. In contrast to more established domains, such as mobile and web-development,
 5921 the distribution of issues are different (see Figure 12.2). Rather than trying to in-
 5922 terpret simple errors (as is the case for CVSs), developers question API usage and
 5923 high-level conceptual questions. Developers have a greater appreciation for the
 5924 technical domain in these mature areas, resulting in fewer shallower questions asked.

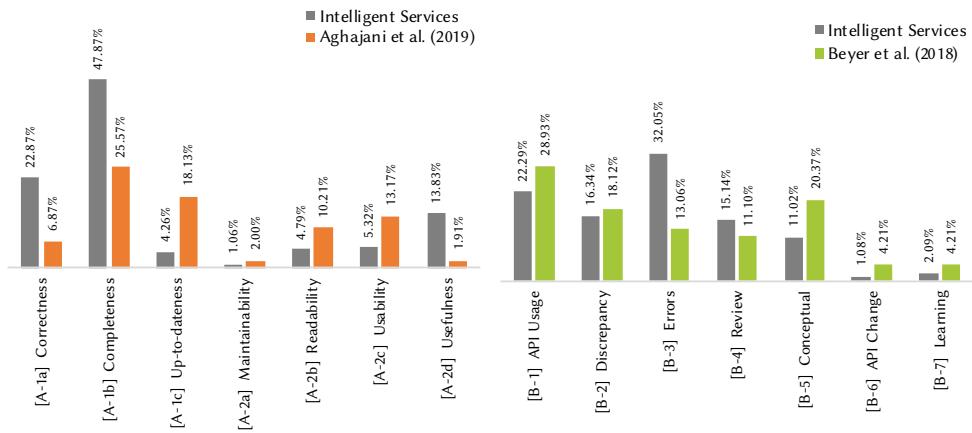


Figure 12.2: The distribution of documentation-specific questions (*left*) and generalised questions (*right*) differs between prior studies. Descriptions of each category for both question types are found in Table 5.1.

5925 **12.1.1.4 RQ4:** “What strategies can developers employ to integrate their applica-
 5926 tions with CVSs while preserving robustness and reliability?”

☞ Developers can employ the use of a facade-based architecture to merge the responses of multiple vendors using a novel, proportional-representation based approach using lexical databases to resolve ontological issues of labels. An integration strategy consisting of four workflows was presented in Chapter 11 to assist developers monitor and handle substantial evolution change in these services. Developers can deal with the probabilistic nature of these services by using a representative data set of their application’s data to fine-tune a confidence threshold and monitor threshold changes in a production setting.

5927 This thesis offers three strategies targeted at improved integration of developer
 5928 applications with CVSs. Chapter 9 successfully demonstrated that multiple services

can be combined using lexical databases to better improve the reliability of relying on a single service's label. Further, this strategy outperformed naive merge methods using a novel proportional representation method by 0.015 F-measure. This strategy uses the idea of a client-server intermediary facade to handle these operations and produce a consistent result regardless of which service is being used. This inspired further work presented in Chapter 11. To handle the evolutionary issues found in the services, we developed a novel integration architecture based on the proxy server strategy, integrating four key proposed workflows which can be used to guard against evolution and non-determinism in these services: (i) initialising a representative benchmark of domain-specific data used in the client application; (ii) validating that the service is behaving as expected against that benchmark; (iii) periodically detecting for evolution if behavioural change occurs, thereby notifying change; and lastly (iv) invalidating future requests if substantial evolution is detected in step (iii). This, in turn, resolves a non-deterministic response into a deterministic error when evolution is raised. Lastly, to deal with the uncertainty arising from probabilistic confidence values, we proposed Threshy (see Chapter 10), a tool to help developers select appropriate threshold boundaries resulting from their benchmark data sets and cost factors (due to missed predictions). Ultimately these strategies aim at improving the robustness of applications that are dependent on CVSs.

12.1.2 Limitations to Research Answers & Future Research

Throughout this thesis, we have used computer vision as a primary exemplar of intelligent AI components provided via the web. Limiting this research to such a narrow scope is an illustrative example that enables more concrete findings and potential solutions to a specific subset of intelligent web services (IWSs). As discussed in Section 1.2, these particular type of IWSs were selected due to both their increasing enthusiasm and uptake in developer communities (see Figure 1.1) 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 [119], many other providers of computer vision intelligence exist [394, 410, 411, 412, 418, 431, 432, 434, 484, 485], including those from Asian market [408, 409, 430, 450, 451] 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

5972 thesis strongly emphasises investigations into identifying issues within web-based
5973 intelligent services. We establish a better understanding on their nature and run-
5974 time behaviour (RQ1), how they are documented (RQ2), and how well they are
5975 understood by developers (RQ3), but only offer limited solutions to these issues
5976 (e.g., RQ4). We encourage the software engineering community to use the issues
5977 identified in this work as a stepping-stone into future solutions, identifying other
5978 ways (beyond improved integration techniques) in which developers can handle
5979 these issues. For example, the broader concepts of our contributed architecture (e.g.,
5980 use of a behaviour token, its parameters, and the error codes proposed) can be shifted
5981 to handle issues in natural language processing to demonstrate the generalisability of
5982 the architecture to other intelligent services, since topic modelling produces labels
5983 with confidences and the approach can be largely transferred to this area.

5984 Other future work stemming from this thesis would be to explore the nature of
5985 other IWSs and understanding if similar evolution and behavioural runtime patterns
5986 exist with their computer vision equivalents (as identified in this thesis). Chiefly,
5987 future work on how to better support developers using different types of intelligent
5988 components would be an interesting area to explore, especially in applying our
5989 design strategies to combat the robustness issues we have identified to these other
5990 types of services and identify any potential pitfalls of our design. As our proposed
5991 architectural usage framework is a preliminary design, rigorous testing in real-
5992 world scenarios, such as a long-term industry case study implementing our design
5993 or conducting formal architecture evaluations such as ATAM [191], would be a
5994 possible avenue of research to verify the design. Further, our proposal makes use
5995 of the benchmark data set approach, but we are yet to explore and test potential
5996 guidelines in developing a benchmark data set. While we provide some potential
5997 guidelines in Section 11.6.3.1, these will need to be evaluated for practical use.

5998 Another key aspect would revolve around the documentation contributions of
5999 this study and investigating whether our suggested documentation improvements
6000 are applicable to these different services. Developing improved documentation and
6001 tooling that better support developers when using these IWSs (and how our proposed
6002 architecture fits in) should be explored.

6003 Moreover, since we find these services to be not yet as matured as traditional
6004 software development domains and—like similar emerging software engineering
6005 domains such as web development in the mid-1990s and early-2000s or mobile
6006 development from the mid-2000s to early-2010s—we suspect there to be substantial
6007 growth in the understanding of how we will use these services and maturity in the
6008 developer’s appreciation of its surrounding technical domain. Therefore, it would be
6009 beneficial to repeat some of the studies within this thesis and assess whether there is
6010 an improved understanding of the phenomena occurring within IWSs and whether
6011 developers have a developed mindset of these services and how they can be used.
6012 Thus, different tools, designs or suggestions may result from repetitional studies 5-10
6013 years in the future. This, therefore, identifies evolution in the *maturity* of intelligent
6014 services, and to highlight whether developers are showing a stronger understanding
6015 of the surrounding technical domain behind these services. We strongly encourage
6016 the software engineering community to explore these in such time to identify maturity

6017 in this emerging domain.

6018 **12.2 Concluding Remarks**

6019 To our knowledge, little prior investigation has been conducted to understand IWSs
6020 via the lenses of software quality—primarily the robustness, reliability of the services
6021 and completeness of its documentation. In this thesis, we have shown that the non-
6022 deterministic and probabilistic properties of computer vision IWSs present non-
6023 trivial impacts to the quality of software that they are integrated with, and it is
6024 pivotal that developers have a greater appreciation of the technical domain behind
6025 the AI techniques that empower such services.

6026 In identifying evolutionary and run-time issues of these services, the ways in
6027 which they are (currently) documented and these issues communicated (or not!), and
6028 analysing how developers perceive these services with a deterministic mindset, we
6029 have shown just how fragile the use of such services (as they stand) are. We strongly
6030 encourage vendors to use suggestions made within this research to improve both
6031 their documentation and their integration strategies so that developers can ensure
6032 more robust applications when using these services. Ultimately, intelligent AI
6033 components are still in a nascent stage, and therefore strongly suggest one message
6034 to eager developers: use with caution and be aware of the consequences!

6035

6036

References

6037

- 6038 [1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving,
6039 M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker,
6040 V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: A system for large-
6041 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.
6042 265–283.
- 6043
- 6044 [2] R. Abdalkareem, E. Shihab, and J. Rilling, “What Do Developers Use the Crowd For?
6045 A Study Using Stack Overflow,” *IEEE Software*, vol. 34, no. 2, pp. 53–60, 2017,
6046 DOI 10.1109/MS.2017.31.
- 6047 [3] E. Aghajani, C. Nagy, G. Bavota, and M. Lanza, “A Large-scale empirical study on linguistic
6048 antipatterns affecting apis,” in *Proceedings of the 34th International Conference on Software
6049 Maintenance and Evolution*. Madrid, Spain: IEEE, September 2018. DOI 10.1109/IC-
6050 SME.2018.00012. ISBN 978-1-53-867870-1 pp. 25–35.
- 6051 [4] E. Aghajani, C. Nagy, O. L. Vega-Marquez, M. Linares-Vasquez, L. Moreno, G. Bavota,
6052 and M. Lanza, “Software Documentation Issues Unveiled,” in *Proceedings of the 41st Interna-
6053 tional Conference on Software Engineering*. Montreal, QC, Canada: IEEE, May 2019.
6054 DOI 10.1109/ICSE.2019.00122. ISBN 978-1-72-810869-8. ISSN 0270-5257 pp. 1199–1210.
- 6055 [5] M. Ahazanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, “Classifying stack
6056 overflow posts on API issues,” in *Proceedings of the 25th International Conference on
6057 Software Analysis, Evolution and Reengineering*. Campobasso, Italy: IEEE, March 2018.
6058 DOI 10.1109/SANER.2018.8330213. ISBN 978-1-53-864969-5 pp. 244–254.
- 6059 [6] R. E. Al-Qutaish, “Quality Models in Software Engineering Literature: An Analytical and
6060 Comparative Study,” *Journal of American Science*, vol. 6, no. 3, pp. 166–175, 2010.
- 6061 [7] H. Allahyari and N. Lavesson, “User-oriented assessment of classification model under-
6062 standability,” in *Proceedings of the 11th Scandinavian Conference on Artificial Intelligence*, vol.
6063 227. Trondheim, Norway: IOS Press, May 2011. DOI 10.3233/978-1-60750-754-3-11.
6064 ISBN 978-1-60-750753-6. ISSN 0922-6389 pp. 11–19.
- 6065 [8] M. Allamanis and C. Sutton, “Why, when, and what: Analyzing stack overflow questions
6066 by topic, type, and code,” in *Proceedings of the 10th IEEE International Working Con-
6067 ference on Mining Software Repositories*. San Francisco, CA, USA: IEEE, May 2013.
6068 DOI 10.1109/MSR.2013.6624004. ISBN 978-1-46-732936-1. ISSN 2160-1852 pp. 53–56.
- 6069 [9] C. O. Alm, D. Roth, and R. Sproat, “Emotions from Text: Machine Learning for Text-Based
6070 Emotion Prediction,” in *Proceedings of the Conference on Human Language Technology and
6071 Empirical Methods in Natural Language Processing*. Vancouver, BC, Canada: Association
6072 for Computational Linguistics, October 2005. DOI 10.3115/1220575.1220648, pp. 579–586.
- 6073 [10] N. Alswaidan and M. E. B. Menai, *A survey of state-of-the-art approaches for emotion recog-*

- 6074 nition in text. Springer, 2020, vol. 62, no. 8, DOI 10.1007/s10115-020-01449-0. ISBN
6075 101-1-50-200144-9
- 6076 [11] J. Alway and C. Calhoun, *Critical Social Theory: Culture, History, and the Challenge of*
6077 *Difference*. American Sociological Association, 1997, vol. 26, no. 1, DOI 10.2307/2076647.
- 6078 [12] S. Aman and S. Szpakowicz, "Identifying Expressions of Emotion in Text," in *Proceedings of*
6079 *the 10th International Conference on Text, Speech and Dialogue*. Pilsen, Czech Republic:
6080 Springer, September 2007. DOI 10.1007/978-3-540-74628-7_27, pp. 196–205.
- 6081 [13] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh, "Modeltracker:
6082 Redesigning performance analysis tools for machine learning," in *Proceedings of the 33rd*
6083 *Annual ACM Conference on Human Factors in Computing Systems*. Seoul, Republic of
6084 Korea: ACM, April 2015. DOI 10.1145/2702123.2702509. ISBN 978-1-45-033145-6 pp.
6085 337–346.
- 6086 [14] K. Arnold, "Programmers are People, Too," *ACM Queue*, vol. 3, no. 5, pp. 54–59, 2005,
6087 DOI 10.1145/1071713.1071731. ISSN 1542-7749
- 6088 [15] M. Arnold, D. Piorkowski, D. Reimer, J. Richards, J. Tsay, K. R. Varshney, R. K. E. Bel-
6089 lamy, M. Hind, S. Houde, S. Mehta, A. Mojsilovic, R. Nair, K. N. Ramamurthy, and
6090 A. Olteanu, "FactSheets: Increasing trust in AI services through supplier's declarations of
6091 conformity," *IBM Journal of Research and Development*, vol. 63, no. 4-5, pp. 6:1 – 6:13, 2019,
6092 DOI 10.1147/JRD.2019.2942288.
- 6093 [16] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges
6094 of deep learning," in *Proceedings of the 44th Euromicro Conference on Software En-*
6095 *gineering and Advanced Applications*. Prague, Czech Republic: IEEE, August 2018.
6096 DOI 10.1109/SEAA.2018.00018. ISBN 978-1-53-867382-9 pp. 50–59.
- 6097 [17] W. R. Ashby and J. R. Pierce, "An Introduction to Cybernetics," *Physics Today*, vol. 10, no. 7,
6098 pp. 34–36, July 1957.
- 6099 [18] L. Aversano, D. Guardabascio, and M. Tortorella, "Analysis of the Documentation of ERP
6100 Software Projects," *Procedia Computer Science*, vol. 121, pp. 423–430, January 2017,
6101 DOI 10.1016/j.procs.2017.11.057. ISSN 1877-0509
- 6102 [19] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K. R. Müller, "How
6103 to explain individual classification decisions," *Journal of Machine Learning Research*, vol. 11,
6104 pp. 1803–1831, 2010. ISSN 1532-4435
- 6105 [20] B. Baesens, C. Mues, M. De Backer, J. Vanthienen, and R. Setiono, "Building intelligent credit
6106 scoring systems using decision tables," in *Proceedings of the 5th International Conference on*
6107 *Enterprise Information Systems*, vol. 2. Angers, France: IEEE, April 2003. DOI 10.1007/1-
6108 4020-2673-0_15. ISBN 9-72-988161-8 pp. 19–25.
- 6109 [21] X. Bai, Y. Wang, G. Dai, W. T. Tsai, and Y. Chen, "A framework for contract-based collab-
6110 orative verification and validation of Web services," in *Proceedings of the 10th International*
6111 *Symposium of Component-Based Software Engineering*. Medford, MA, USA: Springer, July
6112 2007. DOI 10.1007/978-3-540-73551-9_18. ISBN 978-3-54-073550-2. ISSN 0302-9743 pp.
6113 258–273.
- 6114 [22] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in
6115 *Proceedings of the 11th Working Conference on Mining Software Repositories*. Hyderabad,
6116 India: ACM, May 2014. DOI 10.1145/2597073.2597083. ISBN 978-1-45-032863-0 pp.
6117 112–121.
- 6118 [23] K. Ballinger, "Simplicity and Utility, or, Why SOAP Lost," [Online] Available: <http://bit.ly/37vLms0>, December 2014, Accessed: 28 August 2018.
- 6119 [24] O. Baños, M. Damas, H. Pomares, I. Rojas, M. A. Tóth, and O. Amft, "A Benchmark
6120 Dataset to Evaluate Sensor Displacement in Activity Recognition," in *Proceedings of the*
6121 *2012 ACM Conference on Ubiquitous Computing*. Pittsburgh, PA, USA: ACM, 2012.
6122 DOI 10.1145/2370216.2370437. ISBN 9781450312240 pp. 1026–1035.
- 6123 [25] S. Barnett, "Extracting technical domain knowledge to improve software architecture," Ph.D.
6124 dissertation, Swinburne University of Technology, Hawthorn, VIC, Australia, 2018.
- 6125 [26] S. Barnett, R. Vasa, and J. Grundy, "Bootstrapping Mobile App Development," in *Proceedings*
6126 *of the 37th International Conference on Software Engineering*. Florence, Italy: IEEE, May
6127 2015. DOI 10.1109/ICSE.2015.216. ISBN 978-1-47-991934-5. ISSN 0270-5257 pp. 657–660.
- 6128

- 6129 [27] S. Barnett, R. Vasa, and A. Tang, "A Conceptual Model for Architecting Mobile Applications,"
6130 in *Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture*. Montreal,
6131 QC, Canada: IEEE, May 2015. DOI 10.1109/WICSA.2015.28. ISBN 978-1-47-991922-2 pp.
6132 105–114.
- 6133 [28] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis
6134 of topics and trends in Stack Overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp.
6135 619–654, 2014, DOI 10.1007/s10664-012-9231-y. ISSN 1573-7616
- 6136 [29] Y. Baruch, "Response rate in academic studies - A comparative analysis," *Human Relations*,
6137 vol. 52, no. 4, pp. 421–438, 1999, DOI 10.1177/001872679905200401. ISSN 0018-7267
- 6138 [30] O. Barzilay, C. Treude, and A. Zagalsky, "Facilitating crowd sourced software engineering via
6139 stack overflow," in *Finding Source Code on the Web for Remix and Reuse*, 2014, no. 4, pp.
6140 289–308. ISBN 978-1-46-146596-6
- 6141 [31] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-
6142 Wesley, 2003. ISBN 0-32-115495-9
- 6143 [32] E. Batbaatar, M. Li, and K. H. Ryu, "Semantic-Emotion Neural Network for Emotion
6144 Recognition From Text," *IEEE Access*, vol. 7, pp. 111 866–111 878, 2019, DOI 10.1109/ac-
6145 cess.2019.2934529.
- 6146 [33] B. E. Bejnordi, M. Veta, P. J. Van Diest, B. Van Ginneken, N. Karssemeijer, G. Litjens, J. A.
6147 W. M. Van Der Laak, M. Hermsen, Q. F. Manson, M. Balkenhol, O. Geessink, N. Stathonikos,
6148 M. C. R. F. Van Dijk, P. Bult, F. Beca, A. H. Beck, D. Wang, A. Khosla, R. Gargya, H. Ir-
6149 shad, A. Zhong, Q. Dou, Q. Li, H. Chen, H. J. Lin, P. A. Heng, C. Haß, E. Bruni, Q. Wong,
6150 U. Halici, M. Ü. Öner, R. Cetin-Atalay, M. Berseth, V. Khvatkov, A. Vylegzhannin, O. Kraus,
6151 M. Shaban, N. Rajpoot, R. Awan, K. Sirinukunwattana, T. Qaiser, Y. W. Tsang, D. Tellez,
6152 J. Annuscheit, P. Hufnagl, M. Valkonen, K. Kartasalo, L. Latonen, P. Ruusuvuori, K. Li-
6153 limatainen, S. Albarqouni, B. Mungal, A. George, S. Demirci, N. Navab, S. Watanabe, S. Seno,
6154 Y. Takenaka, H. Matsuda, H. A. Phoulady, V. Kovalev, A. Kalinovsky, V. Liauchuk, G. Bueno,
6155 M. M. Fernandez-Carrobles, I. Serrano, O. Deniz, D. Racoceanu, and R. Venâncio, "Diagnostic
6156 assessment of deep learning algorithms for detection of lymph node metastases in women with
6157 breast cancer," *Journal of the American Medical Association*, vol. 318, no. 22, pp. 2199–2210,
6158 December 2017, DOI 10.1001/jama.2017.14585. ISSN 1538-3598
- 6159 [34] R. Bellazzi and B. Zupan, "Predictive data mining in clinical medicine: Current issues and
6160 guidelines," *International Journal of Medical Informatics*, vol. 77, no. 2, pp. 81–97, 2008,
6161 DOI 10.1016/j.ijmedinf.2006.11.006. ISSN 1386-5056
- 6162 [35] A. Ben-David, "Monotonicity Maintenance in Information-Theoretic Machine Learning Algo-
6163 rithms," *Machine Learning*, vol. 19, no. 1, pp. 29–43, 1995, DOI 10.1023/A:1022655006810.
6164 ISSN 1573-0565
- 6165 [36] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifier (URI): Generic
6166 syntax," Tech. Rep., 2004.
- 6167 [37] L. L. Berry, A. Parasuraman, and V. A. Zeithaml, "SERVQUAL: A multiple-item scale for
6168 measuring consumer perceptions of service quality," *Journal of Retailing*, vol. 64, no. 1, pp.
6169 12–40, 1988, DOI 10.1016/S0148-2963(99)00084-3. ISBN 00224359. ISSN 0022-4359
- 6170 [38] J. Besson, "The Business Value of Quality," [Online] Available: <https://ibm.co/2u0UDK0>, June
6171 2004.
- 6172 [39] S. Beyer and M. Pinzger, "A manual categorization of android app development issues on stack
6173 overflow," in *Proceedings of the 30th International Conference on Software Maintenance and
6174 Evolution*. Victoria, BC, Canada: IEEE, September 2014. DOI 10.1109/ICSME.2014.88.
6175 ISBN 978-0-76-955303-0 pp. 531–535.
- 6176 [40] S. Beyer, C. MacHo, M. Pinzger, and M. Di Penta, "Automatically classifying posts into question
6177 categories on stack overflow," in *Proceedings of the 26th International Conference on Program
6178 Comprehension*. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3196321.3196333.
6179 ISBN 978-1-45-035714-2. ISSN 0270-5257 pp. 211–221.
- 6180 [41] J. Biggs and K. Collis, "Evaluating the Quality of Learning: The SOLO Taxonomy (Structure
6181 of the Observed Learning Outcome)," *Management in Education*, vol. 1, no. 4, p. 20, 1987,
6182 DOI 10.1177/089202068700100412. ISBN 0-12-097551-1. ISSN 0892-0206
- 6183 [42] J. P. Bigham, R. S. Kaminsky, R. E. Ladner, O. M. Danielsson, and G. L. Hempton, "WebInSight:
6184 Making web images accessible," in *Proceedings of the 8th International ACM SIGACCESS*

- 6185 *Conference on Computers and Accessibility*. Portland, OR, USA: ACM, October 2006.
6186 DOI 10.1145/1168987.1169018, pp. 181–188.
- 6187 [43] J. P. Bigham, C. M. Prince, and R. E. Ladner, “WebAnywhere,” in *Proceedings of the 2008 International Cross-Disciplinary Conference on Web Accessibility*. Beijing, China: ACM, April 2008. DOI 10.1145/1368044.1368060, pp. 73–82.
- 6190 [44] J. J. Blake, L. P. Maguire, T. M. McGinnity, B. Roche, and L. J. McDaid, “The implementation of
6191 fuzzy systems, neural networks and fuzzy neural networks using FPGAs,” *Information Sciences*,
6192 vol. 112, no. 1-4, pp. 151–168, 1998, DOI 10.1016/S0020-0255(98)10029-4. ISSN 0020-0255
- 6193 [45] B. S. Bloom, *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*, 2nd ed.
6194 Addison-Wesley Longman, 1956. ISBN 978-0-58-228010-6
- 6195 [46] B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” in
6196 *Proceedings of the 2nd International Conference on Software Engineering*. San Francisco,
6197 California, USA: IEEE, October 1976. ISSN 0270-5257 pp. 592–605.
- 6198 [47] B. Boehm and V. R. Basili, “Software defect reduction top 10 list,” *Software Management*, pp.
6199 419–421, 2007, DOI 10.1109/9780470049167.ch12. ISBN 978-0-47-004916-7
- 6200 [48] B. W. Boehm, *Software engineering economics*. Englewood Cliffs, NJ, USA: Prentice-Hall,
6201 1981. ISBN 0-13-822122-7
- 6202 [49] C. Bottomley, “What part writer? What part programmer? A survey of practices
6203 and knowledge used in programmer writing,” in *Proceedings of the 2005 IEEE Interna-*
6204 *tional Professional Communication Conference*. Limerick, Ireland: IEEE, July 2005.
6205 DOI 10.1109/IPCC.2005.1494255, pp. 802–812.
- 6206 [50] P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge*,
6207 3rd ed. Washington, DC, USA: IEEE, 2014. ISBN 978-0-7695-5166-1
- 6208 [51] E. Bouwers and A. van Deursen, “A Lightweight Sanity Check for Implemented Architectures,”
6209 *IEEE Software*, vol. 27, no. 4, pp. 44–50, July 2010, DOI 10.1109/MS.2010.60. ISSN 0740-
6210 7459
- 6211 [52] M. Boyd and N. Wilson, “Just ask Siri? A pilot study comparing smartphone digital assistants
6212 and laptop Google searches for smoking cessation advice,” *PLoS ONE*, vol. 13, no. 3, 2018,
6213 DOI 10.1371/journal.pone.0194811. ISSN 1932-6203
- 6214 [53] O. Boz, “Extracting decision trees from trained neural networks,” in *Proceedings of the 8th ACM
6215 SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton,
6216 AB, Canada: ACM, July 2002. DOI 10.1145/775107.775113, pp. 456–461.
- 6217 [54] H. B. Braiek and F. Khomh, “On Testing Machine Learning Programs,” December 2018.
- 6218 [55] M. Bramer, *Principles of Data Mining*, ser. Undergraduate Topics in Computer Science. Lon-
6219 don, England, UK: Springer, 2016, vol. 180, DOI 10.1007/978-1-4471-7307-6. ISBN 978-1-
6220 44-717306-9
- 6221 [56] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, “Two studies of oppor-
6222 tunistic programming: Interleaving web foraging, learning, and writing code,” in *Proceedings
6223 of the SIGCHI Conference on Human Factors in Computing System*. Boston, MA, USA: ACM,
6224 April 2009. DOI 10.1145/1518701.1518944. ISBN 978-1-60-558247-4 pp. 1589–1598.
- 6225 [57] L. Brathall and M. Jørgensen, “Can you trust a single data source exploratory software engi-
6226 neering case study?” *Empirical Software Engineering*, 2002, DOI 10.1023/A:1014866909191.
6227 ISSN 1382-3256
- 6228 [58] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, “What’s your ML Test Score? A rubric for
6229 ML production systems,” in *Proceedings of the 30th Annual Conference on Neural Information
6230 Processing Systems*. Barcelona, Spain: Curran Associates Inc., December 2016.
- 6231 [59] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*.
6232 New York, NY, USA: CRC press, 1984. DOI 10.1201/9781315139470. ISBN 978-1-35-
6233 146049-1
- 6234 [60] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, “Lessons from applying
6235 the systematic literature review process within the software engineering domain,” *Journal of
6236 Systems and Software*, vol. 80, no. 4, pp. 571–583, April 2007, DOI 10.1016/j.jss.2006.07.009.
6237 ISSN 0164-1212
- 6238 [61] J. Brooke, “SUS-A quick and dirty usability scale,” in *Usability Evaluation in Industry*. Corn-
6239 wall, England, UK: Taylor & Francis Ltd, 1996, ch. 21, pp. 189–194. ISBN 978-0-74-840460-5

- [6240] [62] ——, “SUS: a retrospective,” *Journal of Usability Studies*, vol. 8, no. 2, pp. 29–40, 2013. ISSN 1931-3357
- [6241] [63] 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.
- [6242] [64] 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
- [6243] [65] 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.
- [6244] [66] 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.
- [6245] [67] 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.
- [6246] [68] 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.
- [6247] [69] G. Canfora, “User-side testing of Web Services,” in *Proceedings of the 9th European Conference on Software Maintenance and Reengineering*. Manchester, England, UK: IEEE, March 2005. DOI 10.1109/csmr.2005.57. ISSN 1534-5351 p. 301.
- [6248] [70] G. Canfora and M. Di Penta, “Testing services and service-centric systems: Challenges and opportunities,” *IT Professional*, vol. 8, no. 2, pp. 10–17, 2006, DOI 10.1109/MITP.2006.51. ISSN 1520-9202
- [6249] [71] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, “Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission,” in *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 2015-Augus. Sydney, Australia: ACM, August 2015. DOI 10.1145/2783258.2788613. ISBN 978-1-45-033664-2 pp. 1721–1730.
- [6250] [72] F. Casati, H. Kuno, G. Alonso, and V. Machiraju, *Web Services-Concepts, Architectures and Applications*, 2004. ISBN 978-3-64-207888-0
- [6251] [73] J. P. Cavano and J. A. McCall, “A framework for the measurement of software quality,” in *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues*, vol. 3, no. 5, November 1978, DOI 10.1145/800283.811113, pp. 133–139.
- [6252] [74] C. V. Chambers, D. J. Balaban, B. L. Carlson, and D. M. Grasberger, “The effect of microcomputer-generated reminders on influenza vaccination rates in a university-based family practice center.” *The Journal of the American Board of Family Practice / American Board of Family Practice*, vol. 4, no. 1, pp. 19–26, 1991, DOI 10.3122/jabfm.4.1.19. ISSN 0893-8652
- [6253] [75] J. Cheng and R. Greiner, “Learning bayesian belief network classifiers: Algorithms and system,” in *Proceedings of the 14th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, vol. 2056. Ottawa, ON, Canada: Springer, June 2001. DOI 10.1007/3-540-45153-6_14. ISBN 3-54-042144-0. ISSN 1611-3349 pp. 141–151.
- [6254] [76] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, B. K. Ray, and D. S. Moebus, “Orthogonal Defect Classification—A Concept for In-Process Measurements,” *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 943–956, 1992, DOI 10.1109/32.177364. ISSN 0098-5589
- [6255] [77] E. Choi, N. Yoshida, R. G. Kula, and K. Inoue, “What do practitioners ask about code clone? a preliminary investigation of stack overflow,” in *Proceedings of the 9th International Workshop on Software Clones*, Montreal, QC, Canada, March 2015, DOI 10.1109/IWSC.2015.7069890. ISBN 978-1-46-736914-5 pp. 49–50.
- [6256] [78] D. V. Cicchetti, “Guidelines, Criteria, and Rules of Thumb for Evaluating Normed and Standardized Assessment Instruments in Psychology,” *Psychological Assessment*, vol. 6, no. 4, pp. 284–290, 1994, DOI 10.1037/1040-3590.6.4.284. ISSN 10403590

- [79] Digital, “Case Study: Finding defects earlier yields enormous savings,” [Online] Available: <http://bit.ly/36II2cE>, 2003.
- [80] P. Clark and R. Boswell, “Rule induction with CN2: Some recent improvements,” in *Proceedings of the 1991 European Working Session on Learning*. Porto, Portugal: Springer, March 1991. DOI 10.1007/BFb0017011. ISBN 978-3-54-053816-5. ISSN 1611-3349 pp. 151–163.
- [81] J. Cohen, “A Coefficient of Agreement for Nominal Scales,” *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960, DOI 10.1177/001316446002000104. ISSN 1552-3888
- [82] P. Covington, J. Adams, and E. Sargin, “Deep neural networks for youtube recommendations,” in *Proceedings of the 10th ACM Conference on Recommender Systems*. Boston, MA, USA: ACM, September 2016. DOI 10.1145/2959100.2959190. ISBN 978-1-45-034035-9 pp. 191–198.
- [83] M. W. Craven and J. W. Shavlik, “Extracting tree-structured representations of trained neural networks,” in *Proceedings of the 8th International Conference on Neural Information Processing Systems*, vol. 8. Denver, CO, USA: MIT Press, December 1996. ISBN 978-0-26-220107-0 pp. 24–30.
- [84] J. W. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*, 4th ed. SAGE, 2017. ISBN 860-1-40-429618-5
- [85] P. B. Crosby, *Quality is free: The art of making quality certain*. McGraw-Hill, 1979. ISBN 978-0-07-014512-2
- [86] A. Cummaudo, U. Graetsch, M. Curumsing, S. Barnett, R. Vasa, and J. Grundy, “Manual and Automatic Emotion Analysis of Computer Vision Service Pain-Points,” in *Proceedings of the Sixth International Workshop on Emotion Awareness in Software Engineering*. Virtual Event, USA: IEEE, 2021, In Review.
- [87] 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.
- [88] 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.
- [89] A. Cummaudo, S. Barnett, R. Vasa, and J. Grundy, “Threshy: Supporting Safe Usage of Intelligent Web Services,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event, USA: ACM, November 2020. DOI 10.1145/3368089.3417919, pp. 1645–1649.
- [90] A. Cummaudo, S. Barnett, R. Vasa, J. Grundy, and M. Abdelrazek, “Beware the evolving ‘intelligent’ web service! An integration architecture tactic to guard AI-first components,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event, USA: ACM, November 2020. DOI 10.1145/3368089.3409688, pp. 269–280.
- [91] 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.
- [92] A. Cummaudo, R. Vasa, and J. Grundy, “Requirements of API Documentation: A Case Study into Computer Vision Services,” *IEEE Transactions on Software Engineering*, pp. 1–1, 2020, DOI 10.1109/TSE.2020.3047088.
- [93] M. K. Curumsing, “Emotion-Oriented Requirements Engineering,” Ph.D. dissertation, Swinburne University of Technology, Hawthorn, VIC, Australia, 2017.
- [94] H. da Mota Silveira and L. C. Martini, “How the New Approaches on Cloud Computer Vision can Contribute to Growth of Assistive Technologies to Visually Impaired in the Following Years?” *Journal of Information Systems Engineering & Management*, vol. 2, no. 2, pp. 1–3, 2017, DOI 10.20897/jisem.201709. ISSN 2468-4376

- [95] R. M. Davison, M. G. Martinsons, and N. Kock, "Principles of canonical action research," *Information Systems Journal*, vol. 14, no. 1, pp. 65–86, 2004, DOI 10.1111/j.1365-2575.2004.00162.x. ISSN 1350-1917
- [96] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002, DOI 10.1109/4235.996017. ISSN 1089778X
- [97] K. Dejaeger, F. Goethals, A. Giangreco, L. Mola, and B. Baesens, "Gaining insight into student satisfaction using comprehensible data mining techniques," *European Journal of Operational Research*, vol. 218, no. 2, pp. 548–562, 2012, DOI 10.1016/j.ejor.2011.11.022. ISSN 0377-2217
- [98] I. Dey, *Qualitative Data Analysis: A User-Friendly Guide for Social Scientists*. New York, NY: Routledge, 1993. DOI 10.4324/9780203412497. ISBN 978-0-41-505852-0
- [99] V. Dhar, D. Chou, and F. Provost, "Discovering interesting patterns for investment decision making with GLOWER - A genetic learner overlaid with entropy reduction," *Data Mining and Knowledge Discovery*, vol. 4, no. 4, pp. 69–80, 2000, DOI 10.1023/A:1009848126475. ISSN 1384-5810
- [100] V. Dibia, A. Cox, and J. Weisz, "Designing for Democratization: Introducing Novices to Artificial Intelligence Via Maker Kits," in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. Denver, CO, USA: ACM, May 2017, pp. 381–384.
- [101] M. Doderer, K. Yoon, J. Salinas, and S. Kwek, "Protein subcellular localization prediction using a hybrid of similarity search and Error-Correcting Output Code techniques that produces interpretable results," *In Silico Biology*, vol. 6, no. 5, pp. 419–433, 2006. ISSN 1386-6338
- [102] P. Domingos, "Occam's Two Razors: The Sharp and the Blunt," in *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: AAAI, August 1998. DOI 10.1.1.40.3278, pp. 37–43.
- [103] B. Dorn and M. Guzdial, "Learning on the job: Characterizing the programming knowledge and learning strategies of web designers," in *Proceedings of the 28th ACM Conference on Human Factors in Computing Systems*, vol. 2. Atlanta, GA, USA: ACM, April 2010. DOI 10.1145/1753326.1753430. ISBN 978-1-60-558929-9 pp. 703–712.
- [104] F. Doshi-Velez and B. Kim, "Towards A Rigorous Science of Interpretable Machine Learning," 2017.
- [105] F. Doshi-Velez, M. Kortz, R. Budish, C. Bavitz, S. J. Gershman, D. O'Brien, S. Shieber, J. Waldo, D. Weinberger, and A. Wood, "Accountability of AI Under the Law: The Role of Explanation," *SSRN Electronic Journal*, November 2017, In Press, DOI 10.2139/ssrn.3064761.
- [106] R. G. Dromey, "A model for software product quality," *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 146–162, 1995, DOI 10.1109/32.345830. ISBN 978-1-11-815666-7. ISSN 0098-5589
- [107] C. Drummond and R. C. Holte, "Cost curves: An improved method for visualizing classifier performance," *Machine Learning*, vol. 65, no. 1, pp. 95–130, October 2006, DOI 10.1007/s10994-006-8199-5. ISSN 0885-6125
- [108] T. Durieux, Y. Hamadi, and M. Monperrus, "Fully Automated HTML and Javascript Rewriting for Constructing a Self-Healing Web Proxy," in *Proceedings of the 29th International Symposium on Software Reliability Engineering*. Memphis, TN, USA: IEEE, October 2018. DOI 10.1109/ISSRE.2018.00012. ISSN 1071-9458 pp. 1–12.
- [109] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 11, pp. 285–311. ISBN 978-1-84-800043-8
- [110] P. Ekman, W. V. Friesen, and J. Hager, *Facial Action Coding System: A Technique for the Measurement of Facial Movement*. Palo Alto, CA, USA: Consulting Psychologists Press, 1978.
- [111] W. Elazmeh, S. Matwin, D. O'Sullivan, W. Michalowski, and K. Farion, "Insights from predicting pediatric asthma exacerbations from retrospective clinical data," in *Proceedings of the 22nd Conference on Artificial Intelligence*, vol. WS-07-05. Vancouver, BC, Canada: AAAI, July 2007. ISBN 978-1-57-735332-4 pp. 10–15.

- [112] N. Elgendi and A. Elragal, "Big data analytics: A literature review paper," in *Proceedings of the 10th Industrial Conference on Data Mining*. St. Petersburg, Russia: Springer, July 2014. DOI 10.1007/978-3-319-08976-8_16. ISSN 1611-3349 pp. 214–227.
- [113] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust Physical-World Attacks on Deep Learning Visual Classification," in *Proceedings of the 2017 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, July 2018, DOI 10.1109/CVPR.2018.00175. ISBN 978-1-53-866420-9. ISSN 1063-6919 pp. 1625–1634.
- [114] F. Elder, D. Michie, D. J. Spiegelhalter, and C. C. Taylor, "Machine Learning, Neural, and Statistical Classification." *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 436–438, 1996, DOI 10.2307/2291432. ISBN 978-0-13-106360-0. ISSN 0162-1459
- [115] A. J. Feelders, "Prior knowledge in economic applications of data mining," in *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, vol. 1910. Lyon, France: Springer, September 2000. DOI 10.1007/3-540-45372-5_42. ISBN 978-3-540-40166-9. ISSN 1611-3349 pp. 395–400.
- [116] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.
- [117] I. Finalyson, "Nondeterministic Finite Automata," [Online] Available: <http://bit.ly/319GOF9>, Fredericksburg, VA, USA, 2018.
- [118] J. L. Fleiss, "Measuring nominal scale agreement among many raters," *Psychological Bulletin*, vol. 76, no. 5, pp. 378–382, 1971, DOI 10.1037/h0031619.
- [119] Flexera, "RightScale 2019 State of the Cloud Report from Flexera," [Online] Available: <https://bit.ly/3nigT7E>, Itasca, IL, USA, Tech. Rep., 2019.
- [120] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based verification of Web service compositions," in *Proceedings of the 18th International Conference on Automated Software Engineering*. Linz, Austria: IEEE, September 2004. DOI 10.1109/ase.2003.1240303, pp. 152–161.
- [121] A. A. Freitas, "A critical review of multi-objective optimization in data mining," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 2, p. 77, 2004, DOI 10.1145/1046456.1046467. ISSN 1931-0145
- [122] ———, "Comprehensible classification models," *ACM SIGKDD Explorations Newsletter*, vol. 15, no. 1, pp. 1–10, March 2014, DOI 10.1145/2594473.2594475. ISSN 1931-0145
- [123] A. A. Freitas, D. C. Wieser, and R. Apweiler, "On the importance of comprehensible classification models for protein function prediction," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 7, no. 1, pp. 172–182, 2010, DOI 10.1109/TCBB.2008.47. ISSN 1545-5963
- [124] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, February 2007, DOI 10.1126/science.1136800. ISSN 0036-8075
- [125] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian Network Classifiers," *Machine Learning*, vol. 29, no. 2-3, pp. 131–163, 1997, DOI 10.1002/9780470400531.eorms0099. ISSN 0885-6125
- [126] G. Fung, S. Sandilya, and R. B. Rao, "Rule extraction from linear support vector machines," *Studies in Computational Intelligence*, vol. 80, no. 1, pp. 83–107, 2009, DOI 10.1007/978-3-540-75390-2_4.
- [127] D. Gachechiladze, F. Lanubile, N. Novielli, and A. Serebrenik, "Anger and its direction in collaborative software development," in *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track*, IEEE. Buenos Aires, Argentina: IEEE, May 2017. DOI 10.1109/ICSE-NIER.2017.818, pp. 11–14.
- [128] M. Gamer, J. Lemon, I. Fellows, and P. Singh, "Irr: various coefficients of interrater reliability," *R package version 0.83*, 2010.
- [129] S. K. Garg, S. Versteeg, and R. Buyya, "SMICloud: A framework for comparing and ranking cloud services," in *Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing*. Melbourne, Australia: IEEE, December 2011. DOI 10.1109/UCC.2011.36. ISBN 978-0-76-954592-9 pp. 210–218.
- [130] V. Garousi and M. Felderer, "Experience-based guidelines for effective and efficient data extraction in systematic reviews in software engineering," in *Proceedings of the 21st International*

- 6463 *Conference on Evaluation and Assessment in Software Engineering*, vol. Part F1286. Karl-
6464 skrona, Sweden: ACM, June 2017. DOI 10.1145/3084226.3084238. ISBN 978-1-45-034804-1
6465 pp. 170–179.
- 6466 [131] V. Garousi, M. Felderer, and M. V. Mäntylä, “Guidelines for including grey literature and
6467 conducting multivocal literature reviews in software engineering,” *Information and Software
6468 Technology*, vol. 106, pp. 101–121, 2019, DOI 10.1016/j.infsof.2018.09.006. ISSN 0950-5849
- 6469 [132] D. A. Garvin, “What Does ‘Product Quality’ Really Mean?” *MIT Sloan Management Review*,
6470 vol. 26, no. 1, pp. 25–43, 1984. ISSN 0019-848X
- 6471 [133] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumeé, and K. Craw-
6472 ford, “Datasheets for Datasets,” 2018.
- 6473 [134] R. S. Geiger, N. Varoquaux, C. Mazel-Cabasse, and C. Holdgraf, “The Types, Roles, and
6474 Practices of Documentation in Data Analytics Open Source Software Libraries: A Collaborative
6475 Ethnography of Documentation Work,” *Computer Supported Cooperative Work: CSCW: An
6476 International Journal*, vol. 27, no. 3-6, pp. 767–802, May 2018, DOI 10.1007/s10606-018-
6477 9333-1. ISSN 1573-7551
- 6478 [135] GeoSpatial World, “Mapillary and Amazon Rekognition collaborate to build a parking solution
6479 for US cities through computer vision,” [Online] Available: <http://bit.ly/36AdRmS>, September
6480 2018, Accessed: 25 January 2019.
- 6481 [136] M. Gethsiyal Augusta and T. Kathirvalavakumar, “Reverse engineering the neural networks
6482 for rule extraction in classification problems,” *Neural Processing Letters*, vol. 35, no. 2, pp.
6483 131–150, 2012, DOI 10.1007/s11063-011-9207-8. ISSN 1370-4621
- 6484 [137] D. Ghazi, D. Inkpen, and S. Szpakowicz, “Hierarchical approach to emotion recognition and
6485 classification in texts,” in *Proceedings of the 23rd Canadian Conference on Artificial Intelli-
6486 gence*, vol. 6085 LNAI. Ottawa, ON, Canada: Springer, May 2010. DOI 10.1007/978-3-
6487 642-13059-5_7, pp. 40–50.
- 6488 [138] H. L. Gilmore, “Product conformance cost,” *Quality progress*, vol. 7, no. 5, pp. 16–19, 1974.
- 6489 [139] D. Girardi, N. Novielli, D. Fucci, and F. Lanubile, “Recognizing developers’ emotions while
6490 programming,” *Proceedings - International Conference on Software Engineering*, pp. 666–677,
6491 2020, DOI 10.1145/3377811.3380374. ISBN 978-1-45-037121-6
- 6492 [140] R. L. Glass, I. Vessey, and V. Ramesh, “RESRES: The story behind the paper “Research in
6493 software engineering: An analysis of the literature”,” *Information and Software Technology*,
6494 vol. 51, no. 1, pp. 68–70, 2009, DOI 10.1016/j.infsof.2008.09.015. ISSN 0950-5849
- 6495 [141] M. W. Godfrey and D. M. German, “The past, present, and future of software evolution,” in
6496 *Proceedings of the 2008 Frontiers of Software Maintenance*, Beijing, China, October 2008,
6497 DOI 10.1109/FOSM.2008.4659256. ISBN 978-1-42-442655-3 pp. 129–138.
- 6498 [142] M. W. Godfrey and Q. Tu, “Evolution in open source software: a case study,” in
6499 *Conference on Software Maintenance*. San Jose, CA, USA: IEEE, August 2000.
6500 DOI 10.1109/icsm.2000.883030, pp. 131–142.
- 6501 [143] Google LLC, “Classification: Thresholding | Machine Learning Crash Course,” [Online] Avail-
6502 able: <http://bit.ly/36oMgWb>, 2019, Accessed: 5 February 2020.
- 6503 [144] U. M. Graetsch, A. Cummaudo, M. K. Curumsing, R. Vasa, and J. Grundy, “Using Pre-Trained
6504 Emotion Classification Models against Stack Overflow Questions,” in *Proceedings of the 33rd
6505 International Conference on Advanced Information Systems Engineering*. Melbourne, VIC,
6506 Australia: Springer, 2021, In Review.
- 6507 [145] P. D. Grünwald, *The Minimum Description Length Principle*. MIT press, 2019.
6508 DOI 10.7551/mitpress/4643.001.0001.
- 6509 [146] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao, “MS-Celeb-1M: A Dataset and Benchmark for
6510 Large-Scale Face Recognition,” in *Proceedings of the 16th European Conference on Computer
6511 Vision*. Amsterdam, The Netherlands: Springer, 2016. DOI 10.1007/978-3-319-46487-9_6,
6512 pp. 87–102.
- 6513 [147] M. J. Hadley and H. Marc, “Web Application Description Language,” [Online] Available:
6514 <http://bit.ly/2RXRhQ1>, August 2009.
- 6515 [148] H. A. Haensle, C. Fink, R. Schneiderbauer, F. Toberer, T. Buhl, A. Blum, A. Kalloo, A. Ben
6516 Hadj Hassen, L. Thomas, A. Enk, L. Uhlmann, C. Alt, M. Arenbergerova, R. Bakos, A. Baltzer,
6517 I. Bertlich, A. Blum, T. Bokor-Billmann, J. Bowling, N. Braghierioli, R. Braun, K. Buder-
6518 Bakhaya, T. Buhl, H. Cabo, L. Cabrijan, N. Cevic, A. Classen, D. Deltgen, C. Fink, I. Georgieva,

- 6519 L. E. Hakim-Meibodi, S. Hanner, F. Hartmann, J. Hartmann, G. Haus, E. Hoxha, R. Karls,
6520 H. Koga, J. Kreusch, A. Lallas, P. Majenka, A. Marghoob, C. Massone, L. Mekokishvili,
6521 D. Mestel, V. Meyer, A. Neuberger, K. Nielsen, M. Oliviero, R. Pampena, J. Paoli, E. Pawlik,
6522 B. Rao, A. Rendon, T. Russo, A. Sadek, K. Samhaber, R. Schneiderbauer, A. Schweizer,
6523 F. Toberer, L. Trennheuser, L. Vlahova, A. Wald, J. Winkler, P. Wo'bing, and I. Zalaudek, "Man
6524 against Machine: Diagnostic performance of a deep learning convolutional neural network for
6525 dermoscopic melanoma recognition in comparison to 58 dermatologists," *Annals of Oncology*,
6526 vol. 29, no. 8, pp. 1836–1842, May 2018, DOI 10.1093/annonc/mdy166. ISSN 1569-8041
- 6527 [149] K. A. Hallgren, "Computing Inter-Rater Reliability for Observational Data: An Overview and
6528 Tutorial," *Tutorials in Quantitative Methods for Psychology*, vol. 8, no. 1, pp. 23–34, February
6529 2012, DOI 10.20982/tqmp.08.1.p023. ISSN 1913-4126
- 6530 [150] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in *Pro-
6531 ceedings of the 30th International Conference on Neural Information Processing Systems*.
6532 Barcelona, Spain: Curran Associates Inc., December 2016. DOI 978-1-51-083881-9. ISSN
6533 1049-5258 pp. 3323–3331.
- 6534 [151] M. Hasan, E. Agu, and E. Rundesteiner, "Using Hashtags as Labels for Supervised Learning
6535 of Emotions in Twitter Messages," in *Proceedings of the 2014 ACM SIGKDD Workshop on
6536 Healthcare Informatics*. New York, NY, USA: ACM, August 2014, pp. 187–193.
- 6537 [152] S. Haselbeck, R. Weinreich, G. Buchgeher, and T. Kriechbaum, "Microservice Design Space
6538 Analysis and Decision Documentation: A Case Study on API Management," in *Proceedings
6539 of the 11th International Conference on Service-Oriented Computing and Applications*, Paris,
6540 France, November 2019, DOI 10.1109/SOCA.2018.00008, pp. 1–8.
- 6541 [153] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*, 2nd ed., ser.
6542 Data Mining, Inference, and Prediction. Springer, January 2001.
- 6543 [154] B. Hayete and J. R. Bienkowska, "Gotrees: Predicting go associations from protein domain
6544 composition using decision trees," in *Proceedings of the Pacific Symposium on Biocomput-
6545 ing 2005, PSB 2005*. Hawaii, USA: World Scientific Publishing Company, January 2005.
6546 DOI 10.1142/9789812702456_0013. ISBN 9-81-256046-7 pp. 127–138.
- 6547 [155] A. Head, C. Sadowski, E. Murphy-Hill, and A. Knight, "When not to comment: Questions and
6548 tradeoffs with API documentation for C++ projects," in *Proceedings of the 40th International
6549 Conference on Software Engineering*, ser. questions and tradeoffs with API documentation for
6550 C++ projects. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3180155.3180176.
6551 ISSN 0270-5257 pp. 643–653.
- 6552 [156] R. Heckel and M. Lohmann, "Towards Contract-based Testing of Web Services," *Elec-
6553 tronic Notes in Theoretical Computer Science*, vol. 116, pp. 145–156, January 2005,
6554 DOI 10.1016/j.entcs.2004.02.073. ISSN 1571-0661
- 6555 [157] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie, "Dependency
6556 networks for inference, collaborative filtering, and data visualization," *Journal of Machine
6557 Learning Research*, vol. 1, no. 1, pp. 49–75, 2001, DOI 10.1162/153244301753344614. ISSN
6558 1532-4435
- 6559 [158] M. Henning, "API design matters," *Communications of the ACM*, vol. 52, no. 5, pp. 46–56,
6560 2009, DOI 10.1145/1506409.1506424. ISSN 0001-0782
- 6561 [159] F. Hohman, A. Head, R. Caruana, R. DeLine, and S. M. Drucker, "Gamut: A design probe
6562 to understand how data scientists understand machine learning models," in *Proceedings of the
6563 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow, Scotland, UK:
6564 ACM, May 2019. DOI 10.1145/3290605.3300809. ISBN 978-1-45-035970-2
- 6565 [160] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual Analytics in Deep Learning: An
6566 Interrogative Survey for the Next Frontiers," *IEEE Transactions on Visualization and Computer
6567 Graphics*, vol. 25, no. 8, pp. 2674–2693, 2019, DOI 10.1109/TVCG.2018.2843369. ISSN
6568 1941-0506
- 6569 [161] J. W. Horch, *Practical Guide To Software Quality Management*. Artech House, 2003. ISBN
6570 978-1-58-053604-2
- 6571 [162] H. Hosseini, B. Xiao, and R. Poovendran, "Google's cloud vision API is not robust to noise," in
6572 *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications*.
6573 Cancun, Mexico: IEEE, December 2017. DOI 10.1109/ICMLA.2017.0-172. ISBN 978-1-53-
6574 861417-4 pp. 101–105.

- 6575 [163] D. Hou and L. Mo, "Content categorization of API discussions," in *Proceedings of the 29th International Conference on Software Maintenance*. Eindhoven, Netherlands: IEEE, September 6576 2013. DOI 10.1109/ICSM.2013.17, pp. 60–69.
- 6577 [164] C. Howard, "Introducing Google AI," [Online] Available: <http://bit.ly/2uI6vAr>, May 2018, Accessed: 28 August 2018.
- 6578 [165] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," 6579 *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005, DOI 10.1109/TKDE.2005.50. ISSN 1041-4347
- 6580 [166] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, "An empirical evaluation 6581 of the comprehensibility of decision table, tree and rule based predictive models," *Decision 6582 Support Systems*, vol. 51, no. 1, pp. 141–154, April 2011, DOI 10.1016/j.dss.2010.12.003. 6583 ISSN 0167-9236
- 6584 [167] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," 1990.
- 6585 [168] J. Ingino, *Software Architect's Handbook: Become a Successful Software Architect by Implementing Effective Architecture Concepts*. Birmingham, England, UK: Packt Publishing, Ltd., 6586 2018. ISBN 978-1-78862-406-0
- 6587 [169] International Organization for Standardization, "ISO25010:2011 - Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and 6588 software quality models," 2011.
- 6589 [170] ———, "ISO 8402:1986 Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use," [Online] Available: <http://bit.ly/37SK4HP>, 1986.
- 6590 [171] ———, "ISO 9000:2015 Quality management systems – Fundamentals and vocabulary," [Online] Available: <http://bit.ly/37O4oKo>, 2015.
- 6591 [172] ———, "ISO/IEC 9126. Information technology – Software product quality," November 1999.
- 6592 [173] 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, 6593 March 2018. DOI 10.1007/978-3-319-77712-2_22, pp. 229–239.
- 6594 [174] A. Iyengar, "Supporting Data Analytics Applications Which Utilize Cognitive Services," in 6595 *Proceedings of the 37th International Conference on Distributed Computing Systems*. Atlanta, GA, USA: IEEE, June 2017. DOI 10.1109/ICDCS.2017.172. ISBN 978-1-53-861791-5 pp. 6596 1856–1864.
- 6597 [175] N. Japkowicz and M. Shah, *Evaluating learning algorithms: A classification perspective*. Cambridge University Press, 2011, vol. 9780521196, DOI 10.1017/CBO9780511921803. ISBN 978-0-51-192180-3
- 6598 [176] M. W. M. Jaspers, M. Smeulders, H. Vermeulen, and L. W. Peute, "Effects of clinical decision- 6599 support systems on practitioner performance and patient outcomes: A synthesis of high-quality 6600 systematic review findings," *Journal of the American Medical Informatics Association*, vol. 18, 6601 no. 3, pp. 327–334, 2011, DOI 10.1136/amiajnl-2011-000094. ISSN 1067-5027
- 6602 [177] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and 6603 D. K. Busse, "Improving documentation for eSOA APIs through user studies," in *Proceedings 6604 of the First International Symposium on End User Development*, vol. 5435 LNCS. Siegen, 6605 Germany: Springer, March 2009. DOI 10.1007/978-3-642-00427-8_6. ISSN 0302-9743 pp. 6606 86–105.
- 6607 [178] T. Jiang and A. E. Keating, "AVID: An integrative framework for discovering functional 6608 relationship among proteins," *BMC Bioinformatics*, vol. 6, no. 1, p. 136, 2005, DOI 10.1186/1471- 6609 2105-6-136. ISSN 1471-2105
- 6610 [179] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An Empirical Study of 6611 Model-Agnostic Techniques for Defect Prediction Models," *IEEE Transactions on Software 6612 Engineering*, vol. 5589, no. c, pp. 1–1, 2020, DOI 10.1109/tse.2020.2982385. ISSN 0098-5589
- 6613 [180] B. Jimerson and B. Gregory, "Pivotal Cloud Foundry, Google ML, and Spring," [Online] 6614 Available: <http://bit.ly/2RUBIIL>, San Francisco, CA, USA, December 2017.
- 6615 [181] Y. Jin, *Multi-Objective Machine Learning*, ser. Studies in Computational Intelligence. Berlin, 6616 Heidelberg: Springer, 2006. DOI 10.1007/3-540-33019-4. ISBN 978-3-54-030676-4
- 6617 [182] U. Johansson and L. Niklasson, "Evolving decision trees using oracle guides," in *Proceedings 6618 of the 2009 IEEE Symposium on Computational Intelligence and Data Mining*. Nashville, 6619

- 6630 TN, USA: IEEE, May 2009. DOI 10.1109/CIDM.2009.4938655. ISBN 978-1-42-442765-9
6631 pp. 238–244.
- 6632 [183] M. Jørgensen, T. Dybå, K. Liestøl, and D. I. K. Sjøberg, “Incorrect results in software engineer-
6633 ing experiments: How to improve research practices,” *Journal of Systems and Software*, vol.
6634 116, pp. 133–145, 2016, DOI 10.1016/j.jss.2015.03.065. ISSN 0164-1212
- 6635 [184] J. M. Juran, *Juran on Planning for Quality*. New York, NY, USA: The Free Press, 1988. ISBN
6636 978-0-02-916681-9
- 6637 [185] N. Juristo and O. S. Gómez, “Replication of software engineering experiments,” in *Proceedings*
6638 *of the LASER Summer School on Software Engineering*. Elba Island, Italy: Springer, 2011.
6639 DOI 10.1007/978-3-642-25231-0_2. ISBN 978-3-64-225230-3. ISSN 0302-9743 pp. 60–88.
- 6640 [186] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*. Boston, MA,
6641 USA: Springer, March 2001. DOI 10.1007/978-1-4757-3304-4.
- 6642 [187] D. Kahneman, *Thinking, Fast and Slow*. Macmillan, 2011. ISBN 978-0-37-453355-7
- 6643 [188] A. Karwath and R. D. King, “Homology induction: The use of machine learning to improve se-
6644 quence similarity searches,” *BMC Bioinformatics*, vol. 3, no. 1, p. 11, 2002, DOI 10.1186/1471-
6645 2105-3-11. ISSN 1471-2105
- 6646 [189] K. A. Kaufman and R. S. Michalski, “Learning from inconsistent and noisy data: The AQ18
6647 approach,” in *Proceedings of the 11th European Conference on Principles and Practice of*
6648 *Knowledge Discovery in Databases*, vol. 1609. Warsaw, Poland: Springer, September 1999.
6649 DOI 10.1007/BFb0095128. ISBN 3-540-65965-X. ISSN 1611-3349 pp. 411–419.
- 6650 [190] D. Kavaler, D. Posnett, C. Gibler, H. Chen, P. Devanbu, and V. Filkov, “Using and asking:
6651 APIs used in the Android market and asked about in StackOverflow,” in *Proceedings of the*
6652 *5th International Conference on Social Infomatics*. Kyoto, Japan: Springer, November 2013.
6653 DOI 10.1007/978-3-319-03260-3_35. ISBN 978-3-31-903259-7. ISSN 0302-9743 pp. 405–
6654 418.
- 6655 [191] R. Kazman, M. Klein, and P. Clements, “ATAM: Method for architecture evaluation,” Software
6656 Engineering Institute, Pittsburgh, PA, USA, Tech. Rep., 2000.
- 6657 [192] B. Kim, “Interactive and Interpretable Machine Learning Models for Human Machine Collab-
6658 oration,” Ph.D. dissertation, Massachusetts Institute of Technology, 2015.
- 6659 [193] B. Kim, C. Rudin, and J. Shah, “The Bayesian case model: A generative approach for case-
6660 based reasoning and prototype classification,” in *Proceedings of the 28th Conference on Neural*
6661 *Information Processing Systems*, Montreal, QC, Canada, December 2014. ISSN 1049-5258 pp.
6662 1952–1960.
- 6663 [194] B. Kitchenham and S. Charters, “Guidelines for performing Systematic Literature Reviews
6664 in Software Engineering,” Software Engineering Group, Keele University and Department of
6665 Computer Science, University of Durham, Keele, UK, Tech. Rep., 2007.
- 6666 [195] B. A. Kitchenham and S. L. Pfleeger, “Personal opinion surveys,” in *Guide to Advanced*
6667 *Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer,
6668 November 2007, ch. 3, pp. 63–92. ISBN 978-1-84-800043-8
- 6669 [196] B. A. Kitchenham, T. Dybå, and M. Jorgensen, “Evidence-Based Software Engineering,” in
6670 *Proceedings of the 26th International Conference on Software Engineering*. Edinburgh,
6671 Scotland, UK: IEEE, May 2004. ISBN 978-0-76-952163-3 pp. 273–281.
- 6672 [197] H. K. Klein and M. D. Myers, “A set of principles for conducting and evaluating interpretive
6673 field studies in information systems,” *MIS Quarterly: Management Information Systems*, vol. 23,
6674 no. 1, pp. 67–94, 1999, DOI 10.2307/249410. ISSN 0276-7783
- 6675 [198] A. J. Ko and Y. Riche, “The role of conceptual knowledge in API usability,” in *Proceedings of*
6676 *the 2011 IEEE Symposium on Visual Languages and Human Centric Computing*. Pittsburgh,
6677 PA, USA: IEEE, September 2011. DOI 10.1109/VLHCC.2011.6070395. ISBN 978-1-45-
6678 771245-6 pp. 173–176.
- 6679 [199] A. J. Ko, B. A. Myers, and H. H. Aung, “Six learning barriers in end-user programming
6680 systems,” in *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human*
6681 *Centric Computing*. Rome, Italy: IEEE, September 2004. DOI 10.1109/vlhcc.2004.47.
6682 ISBN 0-78-038696-5 pp. 199–206.
- 6683 [200] I. Kononenko, “Inductive and bayesian learning in medical diagnosis,” *Applied Artificial Intel-
6684 ligence*, vol. 7, no. 4, pp. 317–337, 1993, DOI 10.1080/08839519308949993. ISSN 1087-6545

- 6685 [201] J. Kotula, "Using patterns to create component documentation," *IEEE Software*, vol. 15, no. 2,
6686 pp. 84–92, 1998, DOI 10.1109/52.663791. ISSN 0740-7459
- 6687 [202] S. Krig, "Ground Truth Data, Content, Metrics, and Analysis," in *Computer Vision Metrics: Textbook Edition*. Cham: Springer International Publishing, 2016, pp. 247–271. ISBN 978-3-319-33762-3
- 6689 [203] K. Krippendorff, *Content Analysis*, ser. An Introduction to Its Methodology. SAGE, 1980. ISBN 978-1-50-639566-1
- 6691 [204] 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
- 6693 [205] 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
- 6695 [206] 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.
- 6700 [207] G. Laforge, "Machine Intelligence at Google Scale," in *QCon*, London, England, UK, June 2018.
- 6701 [208] 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.
- 6702 [209] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, p. 159, March 1977, DOI 10.2307/2529310. ISSN 0006-341X
- 6704 [210] 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
- 6705 [211] 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
- 6706 [212] 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.
- 6707 [213] 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
- 6709 [214] 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
- 6710 [215] 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
- 6711 [216] 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.
- 6712 [217] 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.
- 6713 [218] 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.

- [219] 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
- [220] M. Litwin, *How to Measure Survey Reliability and Validity*. Thousand Oaks, CA, USA: SAGE, 1995, vol. 7, DOI 10.4135/9781483348957. ISBN 978-0-80-395704-6
- [221] Y. Liu, T. Kohlberger, M. Norouzi, G. E. Dahl, J. L. Smith, A. Mohtashamian, N. Olson, L. H. Peng, J. D. Hipp, and M. C. Stumpe, "Artificial Intelligence-Based Breast Cancer Nodal Metastasis Detection." *Archives of Pathology & Laboratory Medicine*, vol. 143, no. 7, pp. 859–868, July 2017, DOI 10.5858/arpa.2018-0147-OA. ISSN 1543-2165
- [222] D. Lo Giudice, C. Mines, A. LeClair, R. Curran, and A. Homan, "How AI Will Change Software Development And Applications," [Online] Available: <http://bit.ly/38RiAIN>, Forrester Research, Inc., Tech. Rep., November 2016.
- [223] A. A. Lopez-Lorca, T. Miller, S. Pedell, A. Mendoza, A. Keirnan, and L. Sterling, "One size doesn't fit all: diversifying the user using personas and emotional scenarios," in *Proceedings of the 6th International Workshop on Social Software Engineering*. Hong Kong, China: ACM, November 2014. DOI 10.1145/2661685.2661691, pp. 25–32.
- [224] R. Lori and M. Oded, *Data mining with decision trees*. World Scientific Publishing Company, 2008, vol. 69. ISBN 978-9-81-277171-1
- [225] R. E. Lyons and W. Vanderkulk, "The Use of Triple-Modular Redundancy to Improve Computer Reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, April 2010, DOI 10.1147/rd.62.0200. ISSN 0018-8646
- [226] 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
- [227] G. Malgieri and G. Comandé, "Why a right to legibility of automated decision-making exists in the general data protection regulation," *International Data Privacy Law*, vol. 7, no. 4, pp. 243–265, June 2017, DOI 10.1093/idpl/ixp019. ISSN 2044-4001
- [228] L. Mandel, "Describe REST Web services with WSDL 2.0," [Online] Available: <https://ibm.co/313RoNV>, May 2008, Accessed: 28 August 2018.
- [229] T. E. Marshall and S. L. Lambert, "Cloud-based intelligent accounting applications: Accounting task automation using IBM watson cognitive computing," *Journal of Emerging Technologies in Accounting*, vol. 15, no. 1, pp. 199–215, 2018, DOI 10.2308/jeta-52095. ISSN 1558-7940
- [230] D. Martens, J. Vanthienen, W. Verbeke, and B. Baesens, "Performance of classification models from a user perspective," *Decision Support Systems*, vol. 51, no. 4, pp. 782–793, 2011, DOI 10.1016/j.dss.2011.01.013. ISSN 0167-9236
- [231] P. Mayring, "Mixing Qualitative and Quantitative Methods," in *Mixed Methodology in Psychological Research*. Sense Publishers, 2007, ch. 6, pp. 27–36. ISBN 978-9-07-787473-8
- [232] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in Software Quality: Concept and Definitions of Software Quality," General Electric Company, Griffiss Air Force Base, NY, USA, Tech. Rep. RADC-TR-77-369, November 1977.
- [233] J. McCarthy, "Programs with common sense," in *Proceedings of the Symposium on the Mechanization of Thought Processes*, Cambridge, MA, USA, 1963, pp. 1–15.
- [234] B. McGowen, "Machine learning with Google APIs," [Online] Available: <http://bit.ly/3aUQpo2>, January 2019.
- [235] M. L. McHugh, "Interrater reliability: The kappa statistic," *Biochemia Medica*, vol. 22, no. 3, pp. 276–282, 2012, DOI 10.11613/bm.2012.031. ISSN 1330-0962
- [236] 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
- [237] L. McLeod and S. G. MacDonell, "Factors that affect software systems development project outcomes: A survey of research," *ACM Computing Surveys*, vol. 43, no. 4, p. 24, 2011, DOI 10.1145/1978802.1978803. ISSN 0360-0300
- [238] J. Meltzoff and H. Cooper, *Critical thinking about research: Psychology and related fields*, 2nd ed. American Psychological Association, 2018. DOI 10.1037/0000052-000.
- [239] 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
- [240] T. Mens and S. Demeyer, *Software Evolution*. Berlin, Heidelberg: Springer, 2008. DOI 10.1007/978-3-540-76440-3. ISBN 978-3-54-076439-7

- 6796 [241] T. Mens, S. Demeyer, M. Wermelinger, R. Hirschfeld, S. Ducasse, and M. Jazayeri, “Chal-
6797 lenges in software evolution,” in *Proceedings of the 8th International Workshop on Principles*
6798 *of Software Evolution*, vol. 2005. Lisbon, Portugal: IEEE, September 2005. DOI 10.1109/I-
6799 WPSE.2005.7. ISBN 0-76-952349-8. ISSN 1550-4077 pp. 13–22.
- 6800 [242] A. C. Michalos and H. A. Simon, *The Sciences of the Artificial*. MIT press, 1970, vol. 11,
6801 no. 1, DOI 10.2307/3102825.
- 6802 [243] D. Michie, “Machine learning in the next five years,” in *Proceedings of the 3rd European*
6803 *Conference on European Working Session on Learning*. Glasgow, Scotland, UK: Pitman
6804 Publishing, Inc., October 1988. ISBN 978-0-27-308800-4 pp. 107–122.
- 6805 [244] G. A. Miller, “WordNet: A Lexical Database for English,” *Communications of the ACM*, vol. 38,
6806 no. 11, pp. 39–41, November 1995, DOI 10.1145/219717.219748. ISSN 1557-7317
- 6807 [245] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D.
6808 Raji, and T. Gebru, “Model cards for model reporting,” in *Proceedings of the 2nd Conference*
6809 *on Fairness, Accountability, and Transparency*. Atlanta, GA, USA: ACM, January 2019.
6810 DOI 10.1145/3287560.3287596. ISBN 978-1-45-036125-5 pp. 220–229.
- 6811 [246] R. Mohanani, I. Salman, B. Turhan, P. Rodríguez, and P. Ralph, “Cognitive Biases in Software
6812 Engineering: A Systematic Mapping Study,” *IEEE Transactions on Software Engineering*, p. 1,
6813 2018, DOI 10.1109/TSE.2018.2877759. ISSN 1939-3520
- 6814 [247] D. Moody, “The physics of notations: Toward a scientific basis for constructing visual notations
6815 in software engineering,” *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp.
6816 756–779, 2009, DOI 10.1109/TSE.2009.67. ISSN 0098-5589
- 6817 [248] A. Murgia, P. Tourani, B. Adams, and M. Ortú, “Do developers feel emotions? an ex-
6818 ploratory analysis of emotions in software artifacts,” in *Proceedings of the 11th Work-*
6819 *ing Conference on Mining Software Repositories*. Hyderabad, India: ACM, May 2014.
6820 DOI 10.1145/2597073.2597086, pp. 262–271.
- 6821 [249] C. Murphy and G. Kaiser, “Improving the Dependability of Machine Learning Applications,”
6822 Department of Computer Science, Columbia University, New York, NY, USA, Tech. Rep. MI,
6823 2008.
- 6824 [250] C. Murphy, G. Kaiser, and M. Arias, “An approach to software testing of machine learning
6825 applications,” in *Proceedings of the 19th International Conference on Software Engineering and*
6826 *Knowledge Engineering*. Boston, MA, USA, July 2007. ISBN 978-1-62-748661-3 pp. 167–172.
- 6827 [251] B. A. Myers, A. J. Ko, T. D. LaToza, and Y. Yoon, “Programmers Are Users Too: Human-
6828 Centered Methods for Improving Programming Tools,” *Computer*, vol. 49, no. 7, pp. 44–52,
6829 2016, DOI 10.1109/MC.2016.200.
- 6830 [252] B. A. Myers and J. Stylos, “Improving API Usability,” *Communications of the ACM*, vol. 59,
6831 no. 6, pp. 62–69, May 2016, DOI 10.1145/2896587. ISSN 0001-0782
- 6832 [253] B. A. Myers, S. Y. Jeong, Y. Xie, J. Beaton, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K.
6833 Busse, “Studying the Documentation of an API for Enterprise Service-Oriented Architecture,”
6834 *Journal of Organizational and End User Computing*, vol. 22, no. 1, pp. 23–51, January 2010,
6835 DOI 10.4018/joeuc.2010101903. ISSN 1546-2234
- 6836 [254] C. Myers, A. Furqan, J. Nebolsky, K. Caro, and J. Zhu, “Patterns for how users overcome
6837 obstacles in Voice User Interfaces,” in *Proceedings of the 2018 CHI Conference on Human*
6838 *Factors in Computing Systems*, vol. 2018-April. Montreal, QC, Canada: ACM, April 2018.
6839 DOI 10.1145/3173574.3173580. ISBN 978-1-45-035620-6 p. 6.
- 6840 [255] S. Nakajima, “Model-Checking Verification for Reliable Web Service,” in *Proceedings of the*
6841 *First International Symposium on Cyber World*. Montreal, QC, Canada: IEEE, November
6842 2002. ISBN 978-0-76-951862-6 pp. 378–385.
- 6843 [256] M. Narayanan, E. Chen, J. He, B. Kim, S. Gershman, and F. Doshi-Velez, “How do Humans
6844 Understand Explanations from Machine Learning Systems? An Evaluation of the Human-
6845 Interpretability of Explanation,” *IEEE Transactions on Evolutionary Computation*, 2018, In
6846 Press.
- 6847 [257] S. Narayanan and S. A. McIlraith, “Simulation, verification and automated composition of web
6848 services,” in *Proceedings of the 11th International Conference on World Wide Web*. Honolulu,
6849 HI, USA: ACM, May 2002. DOI 10.1145/511446.511457. ISBN 1-58-113449-5 pp. 77–88.
- 6850 [258] B. J. Nelson, “Remote Procedure Call,” Ph.D. dissertation, Carnegie Mellon University, 1981.

- [259] H. F. Niemeyer and A. C. Niemeyer, "Apportionment methods," *Mathematical Social Sciences*, vol. 56, no. 2, pp. 240–253, 2008. ISSN 0165-4896
- [260] Y. Nishi, S. Masuda, H. Ogawa, and K. Uetsuki, "A test architecture for machine learning product," in *Proceedings of the 11th International Conference on Software Testing, Verification and Validation Workshops*. Västerås, Sweden: IEEE, April 2018. DOI 10.1109/ICSTW.2018.00060. ISBN 978-1-53-866352-3 pp. 273–278.
- [261] N. Novielli, F. Calefato, and F. Lanubile, "The challenges of sentiment detection in the social programmer ecosystem," in *Proceedings of the 7th International Workshop on Social Software Engineering*. Bergamo, Italy: ACM, August 2015. DOI 10.1145/2804381.2804387. ISBN 978-1-45-033818-9 pp. 33–40.
- [262] ———, "A gold standard for emotion annotation in stack overflow," in *Proceedings of the 15th International Conference on Mining Software Repositories*. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3196398.3196453. ISBN 9781450357166 pp. 14–17.
- [263] K. Nybom, A. Ashraf, and I. Porres, "A systematic mapping study on API documentation generation approaches," in *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*. Prague, Czech Republic: IEEE, August 2018. DOI 10.1109/SEAA.2018.00081. ISBN 978-1-53-867382-9 pp. 462–469.
- [264] 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.
- [265] 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.
- [266] Open Software Foundation, "Part 3: DCE Remote Procedure Call (RPC)," in *OSF DCE application development guide: revision 1.0*. Prentice Hall, December 1991.
- [267] N. Oreskes, K. Shrader-Frechette, and K. Belitz, "Verification, validation, and confirmation of numerical models in the earth sciences," *Science*, vol. 263, no. 5147, pp. 641–646, 1994, DOI 10.1126/science.263.5147.641. ISSN 0036-8075
- [268] A. L. M. Ortiz, "Curating Content with Google Machine Learning Application Programming Interfaces," in *EIAPortugal*, July 2017.
- [269] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams, "The emotional side of software developers in JIRA," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ACM. Austin, TX, USA: ACM, May 2016. DOI 10.1145/2901739.2903505, pp. 480–483.
- [270] F. E. B. Otero and A. A. Freitas, "Improving the interpretability of classification rules discovered by an ant colony algorithm: Extended results," in *Evolutionary Computation*, vol. 24, no. 3. ACM, 2016. DOI 10.1162/EVCO_a_00155. ISSN 1530-9304 pp. 385–409.
- [271] A. Pal, S. Chang, and J. A. Konstan, "Evolution of experts in question answering communities," in *Proceedings of the 6th International AAAI Conference on Weblogs and Social Media*. Dublin, Ireland: AAAI, June 2012. ISBN 978-1-57-735556-4 pp. 274–281.
- [272] R. Parekh, "Designing AI at Scale to Power Everyday Life," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Halifax, NS, Canada: ACM, August 2017. DOI 10.1145/3097983.3105815, p. 27.
- [273] 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.
- [274] W. G. Parrott, Ed., *Emotions in Social Psychology: Essential Readings*. Philadelphia: Psychology Press, 2001. ISBN 978-0-86-377682-3
- [275] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proceedings of the 33rd International*

- 6906 *Conference on the Advances of Neural Information Processing Systems.* Vancouver, BC,
6907 Canada: Curran Associates, Inc., December 2019, pp. 8026–8037.
- 6908 [276] K. Patel, J. Fogarty, J. A. Landay, and B. Harrison, “Investigating statistical machine learning
6909 as a tool for software development,” in *Proceedings of the 26th SIGCHI Conference on*
6910 *Human Factors in Computing Systems*, ser. CHI ’08. Florence, Italy: ACM, April 2008.
6911 DOI 10.1145/1357054.1357160. ISBN 978-1-60-558011-1 pp. 667–676.
- 6912 [277] C. Pautasso, O. Zimmermann, and F. Leymann, “RESTful web services vs. “Big” web services:
6913 Making the right architectural decision,” in *Proceedings of the 17th International Conference*
6914 *on World Wide Web.* Beijing, China: ACM, April 2008. DOI 10.1145/1367497.1367606.
6915 ISBN 978-1-60-558085-2
- 6916 [278] M. Pazzani, “Comprehensible knowledge discovery: gaining insight from data,” in *Proceedings*
6917 *of the First Federal Data Mining Conference and Exposition*, Washington, DC, USA, 1997, pp.
6918 73–82.
- 6919 [279] M. J. Pazzani, S. Mani, and W. R. Shankle, “Acceptance of rules generated by machine learning
6920 among medical experts,” *Methods of Information in Medicine*, vol. 40, no. 5, pp. 380–385,
6921 2001, DOI 10.1055/s-0038-1634196. ISSN 0026-1270
- 6922 [280] J. Pearl, “The seven tools of causal inference, with reflections on machine learning,” *Communications*
6923 *of the ACM*, vol. 62, no. 3, pp. 54–60, 2019, DOI 10.1145/3241036. ISSN 1557-7317
- 6924 [281] K. Petersen and C. Gencel, “Worldviews, research methods, and their relationship to validity
6925 in empirical software engineering research,” in *Proceedings of the Joint Conference of the*
6926 *23rd International Workshop on Software Measurement and the 8th International Conference*
6927 *on Software Process and Product Measurement.* Ankara, Turkey: IEEE, October 2013.
6928 DOI 10.1109/IWSM-Mensura.2013.22. ISBN 978-0-76-955078-7 pp. 81–89.
- 6929 [282] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic mapping studies in software engi-
6930 neering,” in *Proceedings of the 12th International Conference on Evaluation and Assessment*
6931 *in Software Engineering, EASE 2008*, 2008, DOI 10.14236/ewic/ease2008.8, pp. 68–77.
- 6932 [283] Z. Pezzementi, T. Tabor, S. Yim, J. K. Chang, B. Drozd, D. Guttendorf, M. Wagner, and
6933 P. Koopman, “Putting Image Manipulations in Context: Robustness Testing for Safe Perception,”
6934 in *Proceedings of the 15th IEEE International Symposium on Safety, Security, and Rescue*
6935 *Robotics.* Philadelphia, PA, USA: IEEE, August 2018. DOI 10.1109/SSRR.2018.8468619.
6936 ISBN 978-1-53-865572-6 pp. 1–8.
- 6937 [284] H. Pham, *System Software Reliability*, 1st ed. Springer, 2000. ISBN 978-1-84-628295-9
- 6938 [285] M. Piccioni, C. A. Furia, and B. Meyer, “An empirical study of API usability,” in *Proceedings*
6939 *of the 13th International Symposium on Empirical Software Engineering and Measurement.*
6940 Baltimore, MD, USA: IEEE, October 2013. DOI 10.1109/ESEM.2013.14. ISSN 1949-3770
6941 pp. 5–14.
- 6942 [286] R. M. Pirsig, *Zen and the art of motorcycle maintenance: An inquiry into values*, 1st ed. HarperTorch,
6943 1974. ISBN 9-780-06-058946-2
- 6944 [287] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, “Data lifecycle challenges in production
6945 machine learning: A survey,” *SIGMOD Record*, 2018, DOI 10.1145/3299887.3299891. ISSN
6946 01635808
- 6947 [288] R. S. Pressman, *Software Engineering: A Practitioner’s Approach*, 8th ed. McGraw-Hill,
6948 2005. ISBN 978-0-07-802212-8
- 6949 [289] D. Pyle, *Data Preparation for Data Mining*, 1st ed. Morgan Kaufmann, 1994. ISBN 978-15-
6950 5-860529-9
- 6951 [290] J. R. Quinlan, “Some elements of machine learning,” in *Proceedings of the 9th International*
6952 *Workshop on Inductive Logic Programming*, vol. 1634. Bled, Slovenia: Springer, June 1999.
6953 DOI 10.1007/3-540-48751-4_3. ISBN 3-54-066109-3. ISSN 1611-3349 pp. 15–18.
- 6954 [291] ———, *C4.5: Programs for machine learning.* San Francisco, CA, USA: Morgan Kauffman,
6955 1993. ISBN 978-1-55-860238-0
- 6956 [292] R Core Team, *R - A Language and Environment for Statistical Computing*, \url{https://www.R-
6957 project.org/}, R Foundation for Statistical Computing, Vienna, Austria, 2020.
- 6958 [293] A. Radford, J. Wu, D. Amodei, D. Amodei, J. Clark, M. Brundage, and I. Sutskever, “GPT2:
6959 Better Language Models and Their Implications,” *OpenAI*, 2019.
- 6960 [294] N. Rama Suri, V. S. Srinivas, and M. Narasimha Murty, “A cooperative game theoretic approach
6961 to prototype selection,” in *Proceedings of the 11th European Conference on Principles and*

- 6962 *Practice of Knowledge Discovery in Databases.* Warsaw, Poland: Springer, September 2007.
6963 DOI 10.1007/978-3-540-74976-9_58. ISBN 978-3-54-074975-2. ISSN 0302-9743 pp. 556–
6964 564.
- 6965 [295] C. Raman Anand; Hoder, *Building Intelligent Apps with Cognitive APIs*, 1st ed. Sebastopol,
6966 CA, USA: O'Reilly Media, Inc., 2019. ISBN 978-1-49-205862-5
- 6967 [296] M. Reboucas, G. Pinto, F. Ebert, W. Torres, A. Serebrenik, and F. Castor, “An Empirical Study
6968 on the Usage of the Swift Programming Language,” in *Proceedings of the 23rd International
6969 Conference on Software Analysis, Evolution, and Reengineering.* Suita, Japan: IEEE, March
6970 2016. DOI 10.1109/saner.2016.66, pp. 634–638.
- 6971 [297] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” in *Proceedings of the 2017
6972 Conference on Computer Vision and Pattern Recognition.* Honolulu, HI, USA: IEEE, July
6973 2017, pp. 6517–6525.
- 6974 [298] A. Reis, D. Paulino, V. Filipe, and J. Barroso, “Using online artificial vision services to assist
6975 the blind - An assessment of Microsoft Cognitive Services and Google Cloud Vision,” *Advances
6976 in Intelligent Systems and Computing*, vol. 746, no. 12, pp. 174–184, 2018, DOI 10.1007/978-
6977 3-319-77712-2_17. ISBN 978-3-31-977711-5. ISSN 2194-5357
- 6978 [299] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why Should I Trust You?: Explaining the Predic-
6979 tions of Any Classifier,” in *Proceedings of the 22nd ACM SIGKDD International Conference
6980 on Knowledge Discovery and Data Mining.* San Francisco, CA, USA: ACM, August 2016.
6981 DOI 2939672.2939778, pp. 1135–1144.
- 6982 [300] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, “MLaaS: Machine learning as a service,”
6983 in *Proceedings of the 14th International Conference on Machine Learning and Applications.*
6984 Miami, FL, USA: IEEE, December 2015. DOI 10.1109/ICMLA.2015.152. ISBN 978-1-50-
6985 900287-0 pp. 896–902.
- 6986 [301] G. Richards, V. J. Rayward-Smith, P. H. Sönksen, S. Carey, and C. Weng, “Data mining for
6987 indicators of early mortality in a database of clinical records,” *Artificial Intelligence in Medicine*,
6988 vol. 22, no. 3, pp. 215–231, 2001, DOI 10.1016/S0933-3657(00)00110-X. ISSN 0933-3657
- 6989 [302] G. Ridgeway, D. Madigan, T. Richardson, and J. O’Kane, “Interpretable Boosted Naïve Bayes
6990 Classification,” in *Proceedings of the 4th International Conference on Knowledge Discovery
6991 and Data Mining.* New York, NY, USA: AAAI, 1998, pp. 101–104.
- 6992 [303] G. Ritzer and E. Guba, “The Paradigm Dialog,” *Canadian Journal of Sociology*, vol. 16, no. 4,
6993 p. 446, 1991, DOI 10.2307/3340973. ISSN 0318-6431
- 6994 [304] M. P. Robillard, “What makes APIs hard to learn? Answers from developers,” *IEEE Software*,
6995 vol. 26, no. 6, pp. 27–34, 2009, DOI 10.1109/MS.2009.193. ISSN 0740-7459
- 6996 [305] M. P. Robillard and R. Deline, “A field study of API learning obstacles,” *Empirical Software
6997 Engineering*, vol. 16, no. 6, pp. 703–732, 2011, DOI 10.1007/s10664-010-9150-8. ISSN 1382-
6998 3256
- 6999 [306] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Geros-
7000 all, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd,
7001 and E. Wong, “On-demand developer documentation,” in *Proceedings of the 33rd IEEE In-
7002 ternational Conference on Software Maintenance and Evolution.* Shanghai, China: IEEE,
7003 September 2017. DOI 10.1109/ICSME.2017.17, pp. 479–483.
- 7004 [307] H. Robinson, J. Segal, and H. Sharp, “Ethnographically-informed empirical studies of soft-
7005 ware practice,” *Information and Software Technology*, vol. 49, no. 6, pp. 540–551, 2007,
7006 DOI 10.1016/j.infsof.2007.02.007. ISSN 0950-5849
- 7007 [308] C. Rosen and E. Shihab, “What are mobile developers asking about? A large scale study
7008 using stack overflow,” *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016,
7009 DOI 10.1007/s10664-015-9379-3. ISSN 1573-7616
- 7010 [309] W. Rosenberry, D. Kenney, and G. Fisher, *Understanding DCE.* O'Reilly & Associates, Inc.,
7011 1992. ISBN 978-1-56-592005-7
- 7012 [310] A. Rosenfeld, R. Zemel, and J. K. Tsotsos, “The Elephant in the Room,” 2018.
- 7013 [311] A. S. Ross, M. C. Hughes, and F. Doshi-Velez, “Right for the right reasons: Training differen-
7014 tiable models by constraining their explanations,” in *Proceedings of the 26th International Joint
7015 Conferences on Artificial Intelligence*, Melbourne, Australia, August 2017, DOI 10.24963/ij-
7016 cai.2017/371. ISBN 978-0-99-924110-3. ISSN 1045-0823 pp. 2662–2670.

- 7017 [312] R. J. Rubey and R. D. Hartwick, "Quantitative measurement of program quality," in *Proceedings of the 1968 23rd ACM National Conference*. Las Vegas, NV, USA: ACM, August 1968. DOI 10.1145/800186.810631. ISBN 978-1-45-037486-6 pp. 671–677.
- 7018 [313] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical techniques for analyzing concurrent and probabilistic systems*, ser. CRM Monograph Series, P. Panangaden and F. van Breugel, Eds. American Mathematical Society, 2004, vol. 23.
- 7019 [314] K. Sailunaz, M. Dhaliwal, J. Rokne, and R. Alhajj, "Emotion detection from text and speech: a survey," *Social Network Analysis and Mining*, vol. 8, no. 1, pp. 1–8, 2018, DOI 10.1007/s13278-018-0505-2.
- 7020 [315] J. Sauro and J. R. Lewis, "When designing usability questionnaires, does it hurt to be positive?" in *Proceedings of the 2011 SIGCHI Conference on Human Factors in Computing Systems*, Vancouver, BC, Canada, May 2011, DOI 10.1145/1978942.1979266, pp. 2215–2223.
- 7021 [316] M. Schwabacher and P. Langley, "Discovering communicable scientific knowledge from spatio-temporal data," in *Proceedings of the 18th International Conference on Machine Learning*. Williamstown, MA, USA: Morgan Kaufmann, June 2001. ISBN 978-1-55-860778-1 pp. 489–496.
- 7022 [317] A. Schwaighofer and N. D. Lawrence, *Dataset shift in machine learning*, J. Quiñonero-Candela and M. Sugiyama, Eds. Cambridge, MA, USA: The MIT Press, 2008. ISBN 978-0-26-217005-5
- 7023 [318] T. A. Schwandt, "Qualitative data analysis: An expanded sourcebook," *Evaluation and Program Planning*, vol. 19, no. 1, pp. 106–107, 1996, DOI 10.1016/0149-7189(96)88232-2. ISSN 0149-7189
- 7024 [319] D. Sculley, M. E. Oney, M. Pohl, B. Spitznagel, J. Hainsworth, and Y. Zhou, "Detecting adversarial advertisements in the wild," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Diego, CA, USA: ACM, August 2011. DOI 10.1145/2020408.2020455, pp. 274–282.
- 7025 [320] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Proceedings of the 28th International Conference on Neural Information Processing Systems*. Montreal, QC, Canada: Curran Associates Inc., December 2015. DOI 10.5555/2969442.2969519. ISSN 1049-5258 pp. 2503–2511.
- 7026 [321] 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
- 7027 [322] 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
- 7028 [323] 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.
- 7029 [324] M. P. Sendak, M. Gao, N. Brajer, and S. Balu, "Presenting machine learning model information to clinical end users with model facts labels," *npj Digital Medicine*, vol. 3, no. 1, p. 41, 2020, DOI 10.1038/s41746-020-0253-3. ISSN 2398-6352
- 7030 [325] 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.
- 7031 [326] 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.
- 7032 [327] 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.
- 7033 [328] 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.

- [329] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. New York, NY, USA: Chapman and Hall/CRC, 2004. DOI 10.4324/9780203489536.
- [330] 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
- [331] 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
- [332] S. Singh, M. T. Ribeiro, and C. Guestrin, “Programs as Black-Box Explanations,” November 2016.
- [333] 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.
- [334] 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
- [335] I. Sommerville, *Software Engineering*, 9th ed. Boston, MA, USA: Addison-Wesley, 2011. ISBN 978-0-13-703515-1
- [336] P. Spector, *Summated Rating Scale Construction*. Newbury Park, CA, USA: SAGE, 1992. DOI 10.4135/9781412986038. ISBN 978-0-80-394341-4
- [337] 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.
- [338] M. A. Storey, L. Singer, B. Cleary, F. F. Filho, and A. Zagalsky, “The (R)evolution of social media in software engineering,” in *Future of Software Engineering Proceedings*. Hyderabad, India: ACM, May 2014. DOI 10.1145/2593882.2593887, pp. 100–116.
- [339] C. Strapparava and A. Valitutti, “WordNet-Affect: an Affective Extension of WordNet,” in *Proceedings of the 4th International Conference on Language Resources and Evaluation*. Lisbon, Portugal: European Language Resources Association (ELRA), May 2004, pp. 1083–1086.
- [340] 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
- [341] 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
- [342] S. Subramanian, L. Inozemtseva, and R. Holmes, “Live API documentation,” in *Proceedings of the 36th International Conference on Software Engineering*. Hyderabad, India: ACM, May 2014. DOI 10.1145/2568225.2568313. ISSN 0270-5257 pp. 643–652.
- [343] S. Sun, W. Pan, and L. L. Wang, “A Comprehensive Review of Effect Size Reporting and Interpreting Practices in Academic Journals in Education and Psychology,” *Journal of Educational Psychology*, vol. 102, no. 4, pp. 989–1004, 2010, DOI 10.1037/a0019507. ISSN 0022-0663
- [344] D. Szafron, P. Lu, R. Greiner, D. S. Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell, A. Fyshe, and D. Meeuwis, “Proteome Analyst: Custom predictions with explanations in a web-based tool for high-throughput proteome annotations,” *Nucleic Acids Research*, vol. 32, 2004, DOI 10.1093/nar/gkh485. ISSN 0305-1048
- [345] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *Proceedings of the 2nd International Conference on Learning Representations*. Banff, AB, Canada: ACM, April 2014.
- [346] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA: IEEE, June 2016. DOI 10.1109/CVPR.2016.308. ISBN 978-1-46-738850-4. ISSN 1063-6919 pp. 2818–2826.
- [347] 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.

- 7129 [348] A. Tahir, A. Yamashita, S. Licorish, J. Dietrich, and S. Counsell, "Can you tell me if it
7130 smells? A study on how developers discuss code smells and anti-patterns in Stack Overflow,"
7131 in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software*
7132 *Engineering*. Christchurch, New Zealand: ACM, June 2018. DOI 10.1145/3210459.3210466.
7133 ISBN 978-1-45-036403-4 pp. 68–78.
- 7134 [349] H. Takagi and C. Asakawa, "Transcoding proxy for nonvisual Web access," in *Proceedings of*
7135 *the 2000 ACM Conference on Assistive Technologies*. Arlington, VA, USA: ACM, November
7136 2000. DOI 10.1145/354324.354371, pp. 164–171.
- 7137 [350] G. Tassey, *The economic impacts of inadequate infrastructure for software testing*. National
7138 Institute of Standards and Technology, September 2002. DOI 10.1080/10438590500197315.
7139 ISBN 978-0-75-672618-8
- 7140 [351] A. Taulavuori, E. Niemelä, and P. Kallio, "Component documentation - A key issue in software
7141 product lines," *Information and Software Technology*, vol. 46, no. 8, pp. 535–546, June 2004,
7142 DOI 10.1016/j.infsof.2003.10.004. ISSN 0950-5849
- 7143 [352] R. S. Taylor, "Question-Negotiation and Information Seeking in Libraries," *College and Re-*
7144 *search Libraries*, vol. 29, no. 3, 1968, DOI 10.5860/crl_29_03_178.
- 7145 [353] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolu-
7146 tion using topic models," *Science of Computer Programming*, vol. 80, pp. 457–479, 2014,
7147 DOI 10.1016/j.scico.2012.08.003. ISSN 0167-6423
- 7148 [354] S. Thrun, "Is Learning The n-th Thing Any Easier Than Learning The First?" in *Proceedings*
7149 *of the 8th International Conference on Neural Information Processing Systems*. Denver, CO,
7150 USA: MIT Press, November 1996. ISSN 1049-5258 p. 7.
- 7151 [355] C. Treude, O. Barzilay, and M. A. Storey, "How do programmers ask and answer questions
7152 on the web?" in *Proceedings of the 33rd International Conference on Software Engineering*.
7153 Honolulu, HI, USA: ACM, May 2011. DOI 10.1145/1985793.1985907. ISBN 978-1-45-
7154 030445-0. ISSN 0270-5257 pp. 804–807.
- 7155 [356] B. Turhan, M. Shepperd, and T. Menzies, "On the dataset shift problem in software en-
7156 gineering prediction models," *Empirical Software Engineering*, vol. 17, pp. 62–74, 2012,
7157 DOI 10.1007/s10664-011-9182-8.
- 7158 [357] A. Tversky and D. Kahneman, "Judgment under uncertainty: Heuristics and biases," *Science*,
7159 vol. 185, no. 4157, pp. 1124–1131, 1974.
- 7160 [358] G. Uddin and F. Khomh, "Automatic Mining of Opinions Expressed About APIs in
7161 Stack Overflow," *IEEE Transactions on Software Engineering*, February 2019, In Press,
7162 DOI 10.1109/TSE.2019.2900245. ISSN 1939-3520
- 7163 [359] G. Uddin and M. P. Robillard, "How API Documentation Fails," *IEEE Software*, vol. 32, no. 4,
7164 pp. 68–75, June 2015, DOI 10.1109/MS.2014.80. ISSN 0740-7459
- 7165 [360] M. Usman, R. Britto, J. Börstler, and E. Mendes, "Taxonomies in software engineering: A
7166 Systematic mapping study and a revised taxonomy development method," *Information and*
7167 *Software Technology*, vol. 85, pp. 43–59, May 2017, DOI 10.1016/j.infsof.2017.01.006. ISSN
7168 0950-5849
- 7169 [361] A. Van Assche and H. Blockeel, "Seeing the forest through the trees learning a comprehensible
7170 model from a first order ensemble," in *Proceedings of the 17th International Conference on*
7171 *Inductive Logic Programming*. Corvallis, OR, USA: Springer, June 2007. DOI 10.1007/978-
7172 3-540-78469-2_26. ISBN 3-54-078468-3. ISSN 0302-9743 pp. 269–279.
- 7173 [362] R. Vasa, "Growth and Change Dynamics in Open Source Software Systems," Ph.D. dissertation,
7174 Swinburne University of Technology, Hawthorn, VIC, Australia, 2010.
- 7175 [363] B. Venners, "Design by Contract: A Conversation with Bertrand Meyer," *Artima Developer*,
7176 2003.
- 7177 [364] W. Verbeke, D. Martens, C. Mues, and B. Baesens, "Building comprehensible customer churn
7178 prediction models with advanced rule induction techniques," *Expert Systems with Applications*,
7179 vol. 38, no. 3, pp. 2354–2364, 2011, DOI 10.1016/j.eswa.2010.08.023. ISSN 0957-4174
- 7180 [365] F. Wachter, Mitterstadt, "EU regulations on algorithmic decision-making and a "right to expla-
7181 nation"," in *Proceedings of the 2016 ICML Workshop on Human Interpretability in Machine*
7182 *Learning*, New York, NY, USA, June 2016, pp. 26–30.
- 7183 [366] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With great training comes great
7184 vulnerability: Practical attacks against transfer learning," in *Proceedings of the 27th USENIX*

- 7185 [367] K. Wang, *Cloud Computing for Machine Learning and Cognitive Applications: A Machine*
7186 *Learning Approach*. Cambridge, MA, USA: MIT Press, 2017. ISBN 978-0-26-203641-2

7187 [368] S. Wang, D. Lo, and L. Jiang, “An empirical study on developer interactions in StackOverflow,”
7188 in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. Coimbra, Portugal:
7189 ACM, March 2013. DOI 10.1145/2480362.2480557, pp. 1019–1024.

7190 [369] W. Wang and M. W. Godfrey, “Detecting API usage obstacles: A study of iOS and android
7191 developer questions,” in *Proceedings of the 10th Working Conference on Mining Software*
7192 *Repositories*. San Francisco, CA, USA: IEEE, May 2013. DOI 10.1109/MSR.2013.6624006.
7193 ISBN 978-1-46-732936-1. ISSN 2160-1852 pp. 61–64.

7194 [370] W. Wang, H. Malik, and M. W. Godfrey, “Recommending Posts concerning API Issues in
7195 Developer Q&A Sites,” in *Proceedings of the 12th Working Conference on Mining Software*
7196 *Repositories*. Florence, Italy: IEEE, May 2015. DOI 10.1109/MSR.2015.28. ISBN 978-0-7695-5594-2. ISSN 2160-1860 pp. 224–234.

7197 [371] R. Watson, “Development and application of a heuristic to assess trends in API documentation,”
7198 in *Proceedings of the 30th ACM International Conference on Design of Communication*.
7199 Seattle, WA, USA: ACM, October 2012. DOI 10.1145/2379057.2379112. ISBN 978-1-45-
7200 031497-8 pp. 295–302.

7201 [372] R. Watson, M. Mark Starnes, J. Jeannot-Schroeder, and J. H. Spyridakis, “API documentation
7202 and software community values: A survey of open-source API documentation,” in *Proceedings*
7203 *of the 31st ACM International Conference on Design of Communication*. Greenville, SC,
7204 USA: ACM, September 2013. DOI 10.1145/2507065.2507076, pp. 165–174.

7205 [373] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform*
7206 *Architecture*. Crawfordsville, IN, USA: Prentice-Hall, 2005. ISBN 0-13-148874-0

7207 [374] G. M. Weiss, “Mining with rarity,” *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp.
7208 7–19, 2004, DOI 10.1145/1007730.1007734. ISSN 1931-0145

7209 [375] D. Wettschereck, D. W. Aha, and T. Mohri, “A Review and Empirical Evaluation of Feature
7210 Weighting Methods for a Class of Lazy Learning Algorithms,” *Artificial Intelligence Review*,
7211 vol. 11, no. 1-5, pp. 273–314, 1997, DOI 10.1007/978-94-017-2053-3_11. ISSN 0269-2821

7212 [376] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viegas, and J. Wilson, “The What-If
7213 Tool: Interactive Probing of Machine Learning Models,” *IEEE Transactions on Visualization*
7214 *and Computer Graphics*, vol. 26, no. 1, pp. 56–65, 2019, DOI 10.1109/tvcg.2019.2934619.
7215 ISSN 1077-2626

7216 [377] H. Wickham, “A Layered grammar of graphics,” *Journal of Computational and Graphical*
7217 *Statistics*, vol. 19, no. 1, pp. 3–28, January 2010, DOI 10.1198/jcgs.2009.07098. ISSN 1061-
7218 8600

7219 [378] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, “Requirements engineering paper classification
7220 and evaluation criteria: a proposal and a discussion,” *Requirements Engineering*, vol. 11,
7221 no. 1, pp. 102–107, March 2006, DOI 10.1007/s00766-005-0021-6. ISSN 0947-3602

7222 [379] Wikipedia Contributors, “List of datasets for machine-learning research — Wikipedia, The
7223 Free Encyclopedia,” [Online] Available: <https://bit.ly/3cZgwLb>, 2020.

7224 [380] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning*
7225 *Tools and Techniques*. Morgan Kaufmann, 2016. DOI 10.1016/c2009-0-19715-5. ISBN
7226 978-0-12-804291-5

7227 [381] C. Wohlin and A. Aurum, “Towards a decision-making structure for selecting a research design
7228 in empirical software engineering,” *Empirical Software Engineering*, vol. 20, no. 6, pp. 1427–
7229 1455, May 2015, DOI 10.1007/s10664-014-9319-7. ISSN 1573-7616

7230 [382] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation*
7231 *in Software Engineering*. Berlin, Heidelberg: Springer, 2012. DOI 10.1007/978-3-64-29044-2. ISBN
7232 978-3-64-229044-2

7233 [383] M. L. Wong and K. S. Leung, *Data Mining Using Grammar Based Genetic Programming and*
7234 *Applications*. Springer, 2002. DOI 10.1007/b116131. ISBN 978-0-79-237746-7

7235 [384] M. R. Wrobel, “Emotions in the software development process,” in *Proceedings of 6th Interna-*
7236 *tional Conference on Human System Interactions*. Sopot, Poland: IEEE, June 2013.
7237 DOI 10.1109/HSI.2013.6577875, pp. 518–523.

- 7241 [385] ——, “The Impact of Lexicon Adaptation on the Emotion Mining from Software Engineering
7242 Artifacts,” *IEEE Access*, 2020, DOI 10.1109/ACCESS.2020.2979148. ISSN 21693536
- 7243 [386] X. Yi and K. J. Kochut, “A CP-nets-based design and verification framework for web services
7244 composition,” in *Proceedings of the 2004 IEEE International Conference on Web Services*. San
7245 Diego, CA, USA: IEEE, July 2004. DOI 10.1109/icws.2004.1314810. ISBN 0-76-952167-3
7246 pp. 756–760.
- 7247 [387] R. K. Yin, *Case study research and applications: Design and methods*, 6th ed. Los Angeles,
7248 CA, USA: SAGE, 2017. ISBN 978-1-50-633616-9
- 7249 [388] J. Zahálka and F. Železný, “An experimental test of Occam’s razor in classification,” *Machine
7250 Learning*, vol. 82, no. 3, pp. 475–481, 2011, DOI 10.1007/s10994-010-5227-2. ISSN 0885-6125
- 7251 [389] J. Zhang and R. Kasturi, “Extraction of Text Objects in Video Documents: Recent Progress,” in
7252 *Proceedings of the 8th International Workshop on Document Analysis Systems*. Nara, Japan:
7253 IEEE, September 2008. DOI 10.1109/das.2008.49, pp. 5–17.
- 7254 [390] X. Zhang, A. S. Ross, A. Caspi, J. Fogarty, and J. O. Wobbrock, “Interaction Proxies for Runtime
7255 Repair and Enhancement of Mobile Application Accessibility,” in *Proceedings of the 2017 CHI
7256 Conference on Human Factors in Computing Systems*, ser. CHI ’17. Denver, CO, USA: ACM,
7257 May 2017. DOI 10.1145/3025453.3025846. ISBN 978-1-4503-4655-9 pp. 6024–6037.
- 7258 [391] J. Zhi, V. Garousi-Yusifoğlu, B. Sun, G. Garousi, S. Shahnewaz, and G. Ruhe, “Cost, benefits and
7259 quality of software development documentation: A systematic mapping,” *Journal of Systems
7260 and Software*, vol. 99, pp. 175–198, 2015, DOI 10.1016/j.jss.2014.09.042. ISSN 0164-1212
- 7261 [392] B. Zupan, J. Demšar, M. W. Kattan, J. R. Beck, and I. Bratko, “Machine learning for survival
7262 analysis: a case study on recurrence of prostate cancer,” *Artificial intelligence in medicine*,
7263 vol. 20, no. 1, pp. 59–75, 2000.
- 7264 [393] M. Zur Muehlen, J. V. Nickerson, and K. D. Swenson, “Developing web services choreography
7265 standards - The case of REST vs. SOAP,” *Decision Support Systems*, vol. 40, no. 1, pp. 9–29,
7266 July 2005, DOI 10.1016/j.dss.2004.04.008. ISSN 0167-9236

7267

7268

List of Online Artefacts

7269

7270 The online artefacts listed below have been downloaded and stored on the Deakin
7271 Research Data Store (RDS) for archival purposes at the following location:

7272 RDS29448-Alex-Cummaudo-PhD/datasets/webrefs

- 7273 [394] Affectiva, Inc., “Home - Affectiva : Affectiva,” <http://bit.ly/36sgbMM>, 2018, accessed: 15
7274 October 2018.
- 7275 [395] Amazon Web Services, Inc., “Detecting Labels in an Image,” <https://amzn.to/2TBNTa>, 2018,
7276 accessed: 28 August 2018.
- 7277 [396] ——, “Detecting Objects and Scenes,” <https://amzn.to/2TDed5V>, 2018, accessed: 28 August
7278 2018.
- 7279 [397] ——, “Amazon Rekognition,” <https://amzn.to/2TyT2BL>, 2018, accessed: 13 September 2018.
- 7280 [398] ——, “Aws release notes,” <https://go.aws/2v0RYjr>, 2019, accessed: 18 March 2019.
- 7281 [399] ——, “Actions - amazon rekognition,” <https://amzn.to/392p3dH>, 2019, accessed: 18 March
7282 2019.
- 7283 [400] ——, “Amazon rekognition | aws machine learning blog,” <https://go.aws/37Q7lKc>, 2019, ac-
7284 cessed: 18 March 2019.
- 7285 [401] ——, “Amazon rekognition image,” <https://go.aws/2ubB6qc>, 2019, accessed: 18 March 2019.
- 7286 [402] ——, “Best practices for sensors, input images, and videos - amazon rekognition,” <https://amzn.to/2uZIW0o>, 2019, accessed: 18 March 2019.
- 7287 [403] ——, “Exercise 1: Detect objects and scenes in an image (console) - amazon rekognition,” <https://amzn.to/36TkLnm>, 2019, accessed: 18 March 2019.
- 7288 [404] ——, “Java (sdk v1) code samples for amazon rekognition - aws code sample,” <https://amzn.to/2ugTle3>, 2019, accessed: 18 March 2019.
- 7289 [405] ——, “Limits in amazon rekognition - amazon rekognition,” <https://amzn.to/2On6n0h>, 2019,
7290 accessed: 18 March 2019.
- 7291 [406] ——, “Step 1: Set up an aws account and create an iam user - amazon rekognition,” <https://amzn.to/2tqW4kI>, 2019, accessed: 18 March 2019.
- 7292 [407] ——, “Troubleshooting amazon rekognition video - amazon rekognition,” <https://amzn.to/3b763fS>, 2019, accessed: 18 March 2019.
- 7293 [408] Beijing Geling Shentong Information Technology Co., Ltd., “DeepGlint,” <http://bit.ly/2uHHdPS>, 2018, accessed: 3 April 2019.
- 7294 [409] Beijing Kuangshi Technology Co., Ltd., “Megvii,” <http://bit.ly/2WJYFzk>, 2018, accessed: 3
7295 April 2019.

- [410] Clarifai, Inc., “Enterprise AI Powered Computer Vision Solutions | Clarifai,” <http://bit.ly/2TB3kSa>, 2018, accessed: 13 September 2018.
- [411] CloudSight, Inc., “Image Recognition API & Visual Search Results | CloudSight AI,” <http://bit.ly/2UmNPCw>, 2018, accessed: 13 September 2018.
- [412] Cognitec Systems GmbH, “The face recognition company - Cognitec,” <http://bit.ly/38VguBB>, 2018, accessed: 15 October 2018.
- [413] A. Cummaudo, <http://bit.ly/2KlyhcD>, 2019, accessed: 27 March 2019.
- [414] ———, <http://bit.ly/2G7saFJ>, 2019, accessed: 27 March 2019.
- [415] ———, <http://bit.ly/2G5ZEEe>, 2019, accessed: 27 March 2019.
- [416] ———, “ICSE 2020 Submission #564 Supplementary Materials,” <http://bit.ly/2Z8zOKW>, 2019.
- [417] ———, <http://bit.ly/2G6ZOeC>, 2019, accessed: 27 March 2019.
- [418] Deep AI, Inc., “DeepAI: The front page of A.I. | DeepAI,” <http://bit.ly/2TBNYgf>, 2018, accessed: 26 September 2018.
- [419] Google LLC, “Best practices for enterprise organizations | documentation | google cloud,” <http://bit.ly/2v0RSs5>, 2019, accessed: 18 March 2019.
- [420] ———, “Detect Labels | Google Cloud Vision API Documentation | Google Cloud,” <http://bit.ly/2TD5kcy>, 2018, accessed: 28 August 2018.
- [421] ———, “Class EntityAnnotation | Google.Cloud.Vision.V1,” <http://bit.ly/2TD5fpg>, 2018, accessed: 28 August 2018.
- [422] ———, “Vision API - Image Content Analysis | Cloud Vision API | Google Cloud,” <http://bit.ly/2TD9mBs>, 2018, accessed: 13 September 2018.
- [423] ———, “Machine learning glossary | google developers,” <http://bit.ly/3b38VdL>, 2019, accessed: 18 March 2019.
- [424] ———, “Open Images Dataset V4,” <http://bit.ly/2Ry2vvF>, 2019, accessed: 9 November 2018.
- [425] ———, “Quickstart: Using client libraries | cloud vision api documentation | google cloud,” <http://bit.ly/2RRMQHG>, 2019, accessed: 18 March 2019.
- [426] ———, “Release notes | cloud vision api documentation | google cloud,” <http://bit.ly/2UipY5J>, 2019, accessed: 18 March 2019.
- [427] ———, “Sample applications | cloud vision api documentation | google cloud,” <http://bit.ly/2SdoB5r>, 2019, accessed: 18 March 2019.
- [428] ———, “Tips & tricks | cloud functions documentation | google cloud,” <http://bit.ly/2GZNc8Z>, 2019, accessed: 18 March 2019.
- [429] ———, “Vision ai | derive image insights via ml | cloud vision api | google cloud,” <http://bit.ly/31nWoNx>, 2019, accessed: 18 March 2019.
- [430] Guangzhou Tup Network Technology, “TupuTech,” <http://bit.ly/2uF4IsN>, 2018, accessed: 3 April 2019.
- [431] Imagga Technologies, “Imagga - powerful image recognition APIs for automated categorization & tagging in the cloud and on-premises,” <http://bit.ly/2TxsyRe>, 2018, accessed: 13 September 2018.
- [432] International Business Machines Corporation, “Watson Visual Recognition - Overview | IBM,” <https://ibm.co/2TBNIO4>, 2018, accessed: 13 September 2018.
- [433] ———, “Watson Tone Analyzer,” <https://ibm.co/37w3y4A>, 2019, accessed: 25 January 2019.
- [434] Kairos AR, Inc., “Kairos: Serving Businesses with Face Recognition,” <http://bit.ly/30WHGNs>, 2018, accessed: 15 October 2018.
- [435] Microsoft Corporation, “azure-sdk-for-java/ImageTag.java,” <http://bit.ly/38IDPWU>, 2018, accessed: 28 August 2018.
- [436] ———, “Image Processing with the Computer Vision API | Microsoft Azure,” <http://bit.ly/2YqhkS6>, 2018, accessed: 13 September 2018.
- [437] ———, “How to call the Computer Vision API,” <http://bit.ly/2TD5oJk>, 2018, accessed: 28 August 2018.
- [438] ———, “What is Computer Vision?” <http://bit.ly/2TDgUnU>, 2018, accessed: 28 August 2018.
- [439] ———, “Call the computer vision api - azure cognitive services | microsoft docs,” <http://bit.ly/2vHSdjT>, 2019, accessed: 18 March 2019.
- [440] ———, “Content tags - computer vision - azure cognitive services | microsoft docs,” <http://bit.ly/2vESzHX>, 2019, accessed: 18 March 2019.

- 7357 [441] ——, “Github - azure-samples/cognitive-services-java-computer-vision-tutorial: This tutorial shows the features of the microsoft cognitive services computer vision rest api.” <http://bit.ly/37N1yoN>, 2019, accessed: 18 March 2019.
- 7358 [442] ——, “Improving your classifier - custom vision service - azure cognitive services | microsoft docs,” <http://bit.ly/37SBkRQ>, 2019, accessed: 18 March 2019.
- 7359 [443] ——, “Microsoft azure legal information | microsoft azure,” <https://bit.ly/2Cy8Z8r>, 2019, accessed: 18 March 2019.
- 7360 [444] ——, “Quickstart: Computer vision client library for .net - azure cognitive services | microsoft docs,” <http://bit.ly/2vF3wJC>, 2019, accessed: 18 March 2019.
- 7361 [445] ——, “Release notes - custom vision service - azure cognitive services | microsoft docs,” <http://bit.ly/2UIPiaW>, 2019, accessed: 18 March 2019.
- 7362 [446] ——, “Sample: Explore an image processing app in c# - azure cognitive services | microsoft docs,” <http://bit.ly/2u4mPMh>, 2019, accessed: 18 March 2019.
- 7363 [447] ——, “Tutorial: Generate metadata for azure images - azure cognitive services | microsoft docs,” <http://bit.ly/2RRnARK>, 2019, accessed: 18 March 2019.
- 7364 [448] ——, “Tutorial: Use custom logo detector to recognize azure services - custom vision - azure cognitive services | microsoft docs,” <http://bit.ly/2RUGwPH>, 2019, accessed: 18 March 2019.
- 7365 [449] ——, “What is computer vision? - computer vision - azure cognitive services | microsoft docs,” <http://bit.ly/37SomDx>, 2019, accessed: 18 March 2019.
- 7366 [450] SenseTime, “SenseTime,” <http://bit.ly/2WH6RjF>, 2018, accessed: 3 April 2019.
- 7367 [451] Shanghai Yitu Technology Co., Ltd., “Yitu Technology,” <http://bit.ly/2uGvxgf>, 2018, accessed: 3 April 2019.
- 7368 [452] Stack Overflow User #1008563 ‘samiles’, “AWS Rekognition PHP SDK gives invalid image encoding error,” <http://bit.ly/31Sgpec>, 2019, accessed: 22 June 2019.
- 7369 [453] Stack Overflow User #10318601 ‘reza naderii’, “google cloud vision category detecting,” <http://bit.ly/31Uf32t>, 2019, accessed: 22 June 2019.
- 7370 [454] Stack Overflow User #10729564 ‘gabgob’, “Multiple Google Vision OCR requests at once?” <http://bit.ly/31P09dU>, 2019, accessed: 22 June 2019.
- 7371 [455] Stack Overflow User #1453704 ‘deoptimancode’, “Human body part detection in Android,” <http://bit.ly/31T5pxd>, 2019, accessed: 22 June 2019.
- 7372 [456] Stack Overflow User #174602 ‘geekyaleks’, “aws Rekognition not initializing on iOS,” <http://bit.ly/31UeqG9>, 2019, accessed: 22 June 2019.
- 7373 [457] Stack Overflow User #2251258 ‘James Dorfman’, “All GoogleVision label possibilities?” <http://bit.ly/31R4FZi>, 2019, accessed: 22 June 2019.
- 7374 [458] Stack Overflow User #2521469 ‘Hillary Sanders’, “Is there a full list of potential labels that Google’s Vision API will return?” <http://bit.ly/2KNnJSB>, 2019, accessed: 22 June 2019.
- 7375 [459] Stack Overflow User #2604150 ‘user2604150’, “Google Vision Accent Character Set NodeJs,” <http://bit.ly/31TsVdp>, 2019, accessed: 22 June 2019.
- 7376 [460] Stack Overflow User #3092947 ‘Mark Bench’, “Google Cloud Vision OCR API returning incorrect values for bounding box/vertices,” <http://bit.ly/31UeZjf>, 2019, accessed: 22 June 2019.
- 7377 [461] Stack Overflow User #3565255 ‘CSquare’, “Vision API topicality and score always the same,” <http://bit.ly/2TD5As2>, 2019, accessed: 22 June 2019.
- 7378 [462] Stack Overflow User #4748115 ‘Latifa Al-jiffry’, “similar face recognition using google cloud vision API in android studio,” <http://bit.ly/31WhMZY>, 2019, accessed: 22 June 2019.
- 7379 [463] Stack Overflow User #4852910 ‘Gaurav Mathur’, “Amazon Rekognition Image caption,” <http://bit.ly/31P08qm>, 2019, accessed: 22 June 2019.
- 7380 [464] Stack Overflow User #5294761 ‘Eury Pérez Beltré’, “Specify language for response in Google Cloud Vision API,” <http://bit.ly/31SsUGG>, 2019, accessed: 22 June 2019.
- 7381 [465] Stack Overflow User #549312 ‘GroovyDotCom’, “Image Selection for Training Visual Recognition,” <http://bit.ly/31W8lcw>, 2019, accessed: 22 June 2019.
- 7382 [466] Stack Overflow User #5809351 ‘J.Doe’, “How to confidently validate object detection results returned from Google Cloud Vision,” <http://bit.ly/31UcCNy>, 2019, accessed: 22 June 2019.
- 7383 [467] Stack Overflow User #5844927 ‘Amit Pawar’, “Google cloud Vision and Clarifai doesn’t Support tagging for 360 degree images and videos,” <http://bit.ly/31StuEm>, 2019, accessed: 22 June 2019.

- [468] Stack Overflow User #5924523 ‘Akash Dathan’, “Can i give aspect ratio in Google Vision api?”
http://bit.ly/2KSJwsp, 2019, accessed: 22 June 2019.
- [469] Stack Overflow User #6210900 ‘Mike Grommet’, “Are the Cloud Vision API limits in documentation correct?” http://bit.ly/31SsNLg, 2019, accessed: 22 June 2019.
- [470] 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.
- [471] Stack Overflow User #6841211 ‘NigelJL’, “Google Cloud Vision - Numbers and Numerals OCR,” http://bit.ly/31P07mi, 2019, accessed: 22 June 2019.
- [472] 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.
- [473] Stack Overflow User #7187987 ‘tuanars10’, “Adding a local path to Microsoft Face API by Python,” http://bit.ly/2KLeMt3, 2019, accessed: 22 June 2019.
- [474] Stack Overflow User #7219743 ‘Davide Biraghi’, “Google Vision API does not recognize single digits,” http://bit.ly/31Ws1Nj, 2019, accessed: 22 June 2019.
- [475] Stack Overflow User #738248 ‘lavuy’, “Meaning of score in Microsoft Cognitive Service’s Entity Linking API,” http://bit.ly/2TD9vVw, 2019, accessed: 22 June 2019.
- [476] Stack Overflow User #7604576 ‘Alagappan Narayanan’, “Text extraction - line-by-line,” http://bit.ly/31Yc21s, 2019, accessed: 22 June 2019.
- [477] Stack Overflow User #7692297 ‘1lucas’, “Can Google Cloud Vision generate labels in Spanish via its API?” http://bit.ly/31UcBsY, 2019, accessed: 22 June 2019.
- [478] Stack Overflow User #7896427 ‘David mark’, “Google Api Vision, ““before_request”” error,” http://bit.ly/31Z27Zt, 2019, accessed: 22 June 2019.
- [479] Stack Overflow User #8210103 ‘Cosmin-Ioan Leferman’, “Google Vision API text detection strange behaviour - Javascript,” http://bit.ly/31Ucyxi, 2019, accessed: 22 June 2019.
- [480] 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.
- [481] 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.
- [482] 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.
- [483] Stack Overflow User #9913535 ‘Sahil Mehra’, “Google Vision API: ModuleNotFoundError: module not found ‘google.oauth2’,” http://bit.ly/31VIZfU, 2019, accessed: 22 June 2019.
- [484] Symisc Systems, S.U.A.R.L, “Computer Vision & Media Processing APIs | PixLab,” http://bit.ly/2UlkW9K, 2018, accessed: 13 September 2018.
- [485] Talkwalker Inc., “Image Recognition - Talkwalker,” http://bit.ly/2TyT7W5, 2018, accessed: 13 September 2018.
- [486] 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 [309]. This framework used the synchronous communication paradigm remote procedure calls (RPCs) first introduced by Nelson [258] that allows procedures to be called in a remote address space as if it were local. Its communication paradigm, DCE/RPC [266], 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 [266]). As new functionality was introduced, this lead to the natural development of the Simple Object Access Protocol (SOAP), the backbone messaging connector for web service applications, a realisation of the service-oriented architecture (SOA) [72] pattern. The SOA pattern prescribes that services are offered by service providers and consumed by service consumers in a platform- and language-agnostic manner and are used in large-scale enterprise systems (e.g., banking, health). Key to the SOA pattern is that a service’s quality attributes (see Section 2.1) can be specified and guaranteed using a service-level agreement (SLA) whereby the consumer and provider agree upon a set level of service, which in some cases are legally binding [31]. This agreement can be measured using quality of service (QoS) parameters met by the service provider during the transportation layer (e.g., response time, cost of leasing resources, reliability guarantees, system availability and trust/security assurance [367, 373]). These attributes are included within SOAP headers; thus, QoS aspects are independent from the transport layer and instead exist at the application layer [277]. 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 [277, 393] 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).



Figure A.1: Worldwide search interest for SOAP (blue) and REST (red) since 2004. Source: Google Trends.

Listing A.1: A SOAP HTTP POST consumer request to retrieve customer record #43456 from a web service provider. Source: [23].

```
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: [23].

```
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 [277]. That is, the HTTP is considered as a transport protocol solely. In 2000, Fielding [116] introduced REpresentational State Transfer (REST), which instead approaches the web as a medium to publish data (i.e., HTTP is part of the *application* layer instead). Hence, applications become amalgamated into of the Web. Fielding bases REST on four key principles:

- **URIs identify resources.** Resources and services have a consistent global address space that aides in their discovery via URIs [36].
- **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 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) [228]). Therefore, no RESTful service uses a standardised response document or invocation syntax. While there are proposals, such as WADL [147], 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: [23].

```
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 (p257)** highlights potential causal factors that may influence a developer’s understanding of the documentation and response of IWSs. It was intended to be used as the basis of a survey study in Chapter 8, and can be used for future avenues of research.
- **Figure A.3 (p258)** was intended for the discussion in Chapter 5, where we propose that developers have a misaligned of the technical domain models within IWSs and more specifically CVSs. We designed a draft technical domain model to describe the various aspects developers must consider when using these services, based on the work by Barnett [25].
- **Figure A.4 (p259)** describes potential questions that may arise to analyse and test the causal factors of the technical domain model proposed in Figure A.3. This lies an open avenue of future research.
- **Figure A.5 (p259)** emphasises dichotomy between an application using an IWS and the IWS’ training data (which is sourced from an unknown context) and the context of an application, which is known. This is to emphasise how the model produced from these services need to be calibrated to the application domain being used in order for the decision boundary of a single inference to be properly assessed by the developer. This image was originally included within the Threshy publication (Chapter 10) but was removed due to space limitations.
- **Figure A.6 (p260)** illustrates the domain model of Threshy (Chapter 10).
- **Figure A.7 (p260)** illustrates the dynamic model of using Threshy and its interactions between the application, front-end of Threshy and back-end of Threshy (Chapter 10).
- **Figure A.8 (p261)** 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.9 (p262)** is a class diagram of the reference architecture of the proposed architecture in Chapter 11. The implementation is provided in Chapter B. See Chapter 11 for more.
- **Figure A.10 (p263)** is a sequence diagram illustrating how the reference architecture can be used to create a new benchmark as per the implementation provided in Chapter B. See Chapter 11 for more.
- **Figure A.11 (p264)** is a sequence diagram illustrating how applications can make requests to the proxy server ‘facade’ as per the implementation provided in Chapter B. See Chapter 11 for more.
- **Figure A.12 (p265)** is a state diagram that illustrates the overall states that exist within the architecture tactic’s workflows. See Chapter 11 for more.
- **Figure A.13 (p266)** is a sequence diagram illustrating how the reference architecture handles evolution in an external service per the implementation provided in Chapter B. See Chapter 11 for more.

- **Figure A.14 (p267)** illustrates how the reference architecture is able to capture and handle three requests (two valid, one invalid) when sent to the proxy server. See Chapter 11 for more.

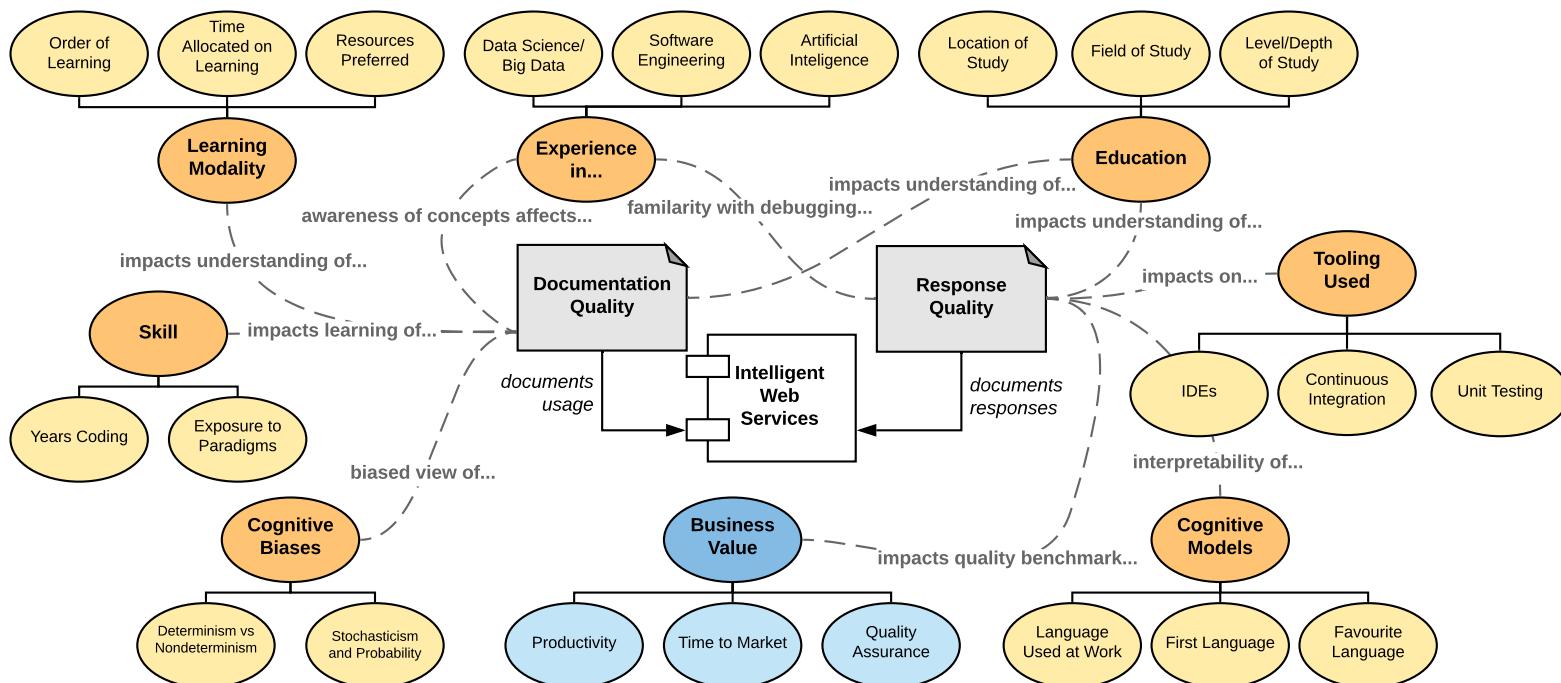
Figure A.2: Causal factors that may influence understanding of intelligent web services.

Figure A.3: A proposal technical domain model for intelligent services. (The  symbol indicates computer vision related services only.)

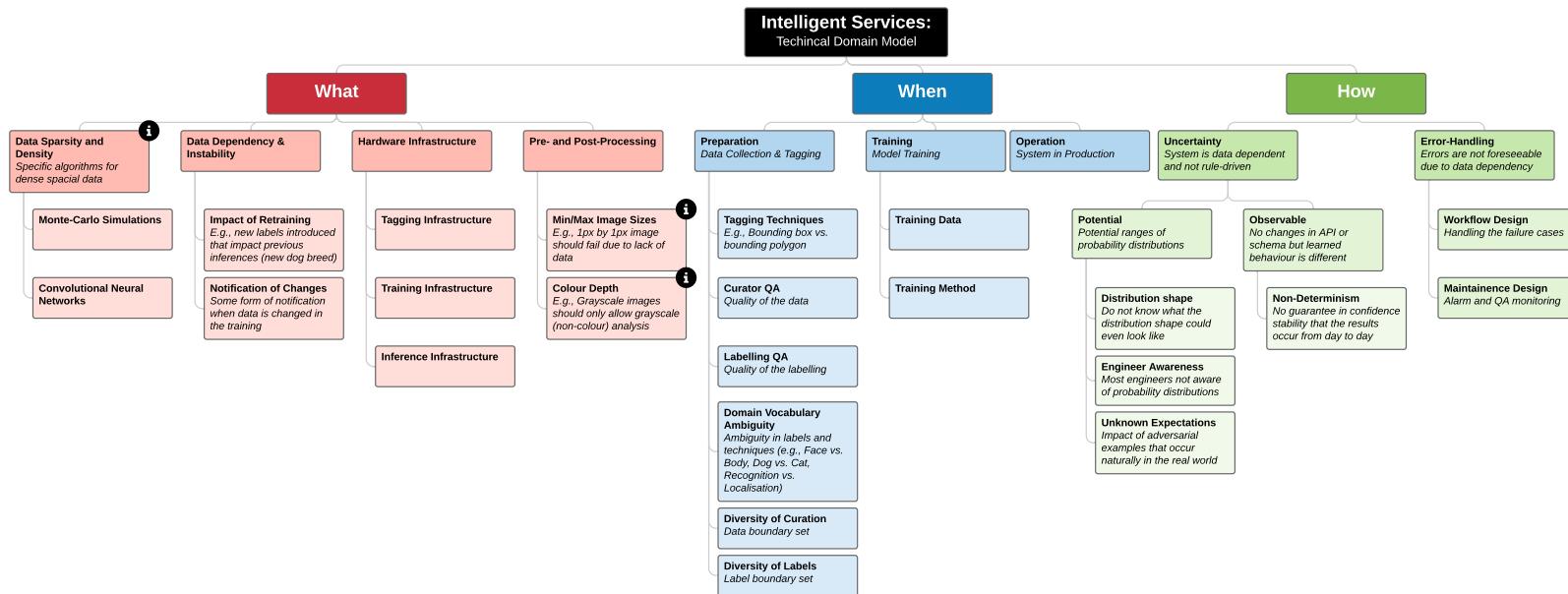


Figure A.4: Potential questions that can be asked around causal factors of a developer's understanding of an intelligent service.

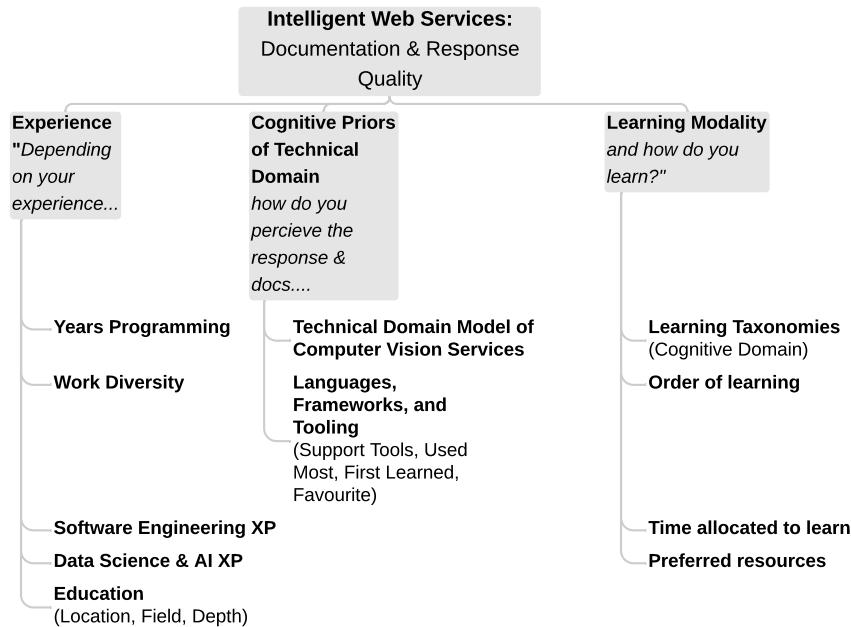


Figure A.5: Threshy assists with making appropriate decision boundaries in the application context by calibrating model (train on an unknown context) to your domain.

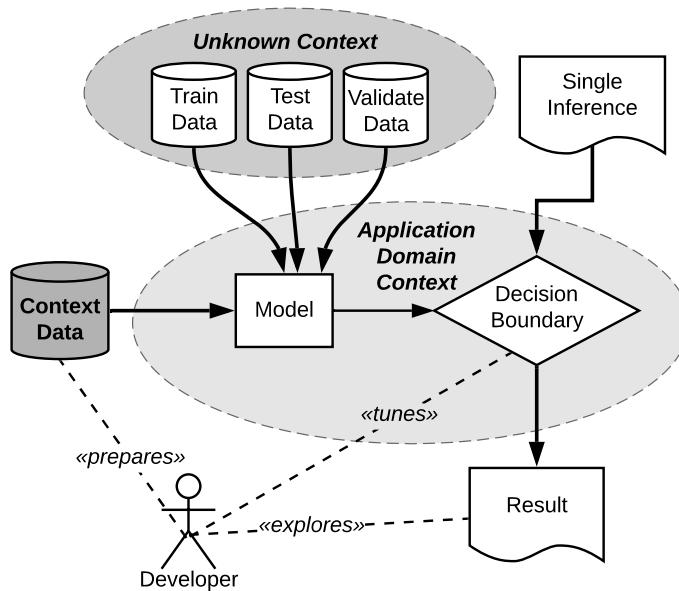


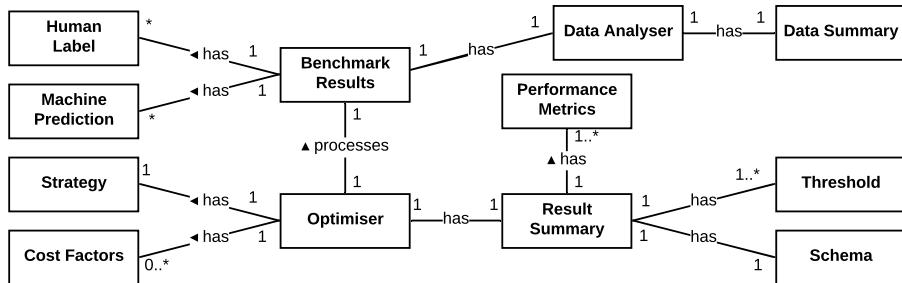
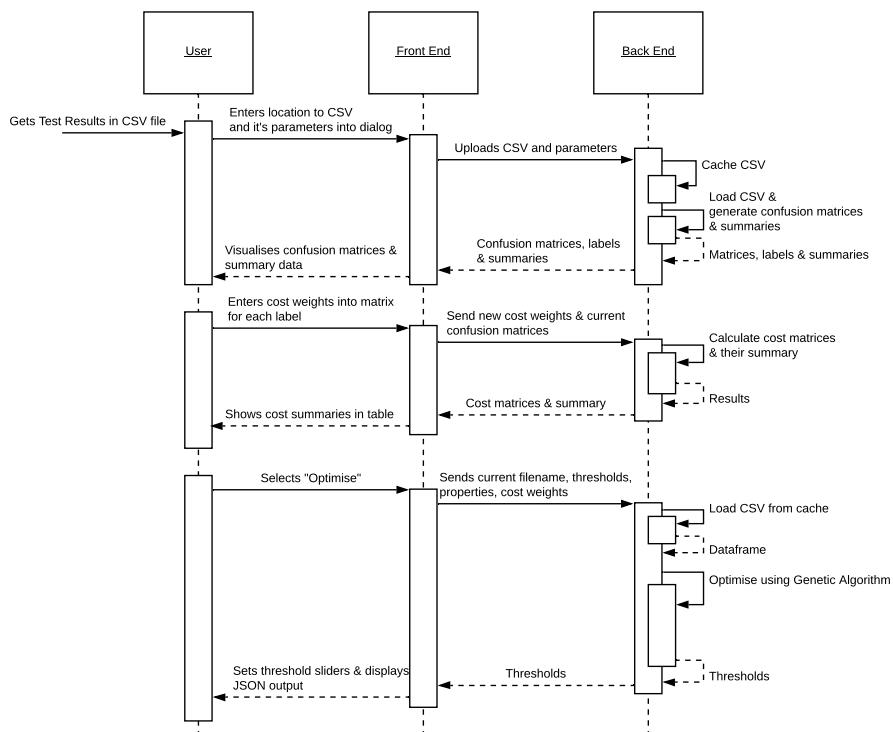
Figure A.6: Threshy domain model.**Figure A.7:** High level overview of Threshy's interaction between the front- and back-end.

Figure A.8: High-level overview of the methodology within Chapter 5.

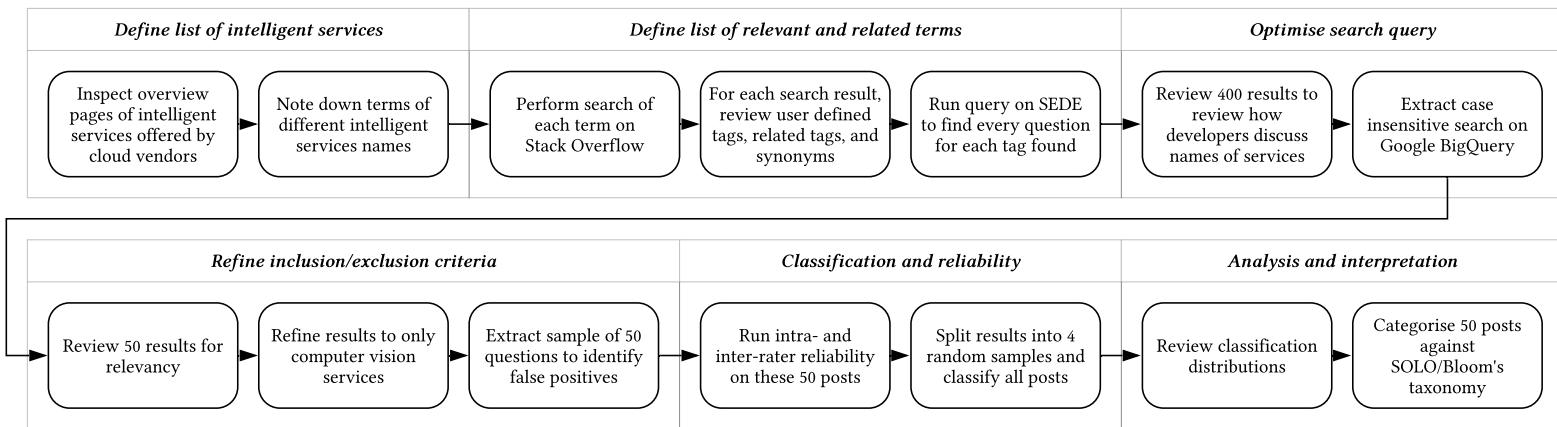


Figure A.9: Class diagram of the implementation of our architecture.

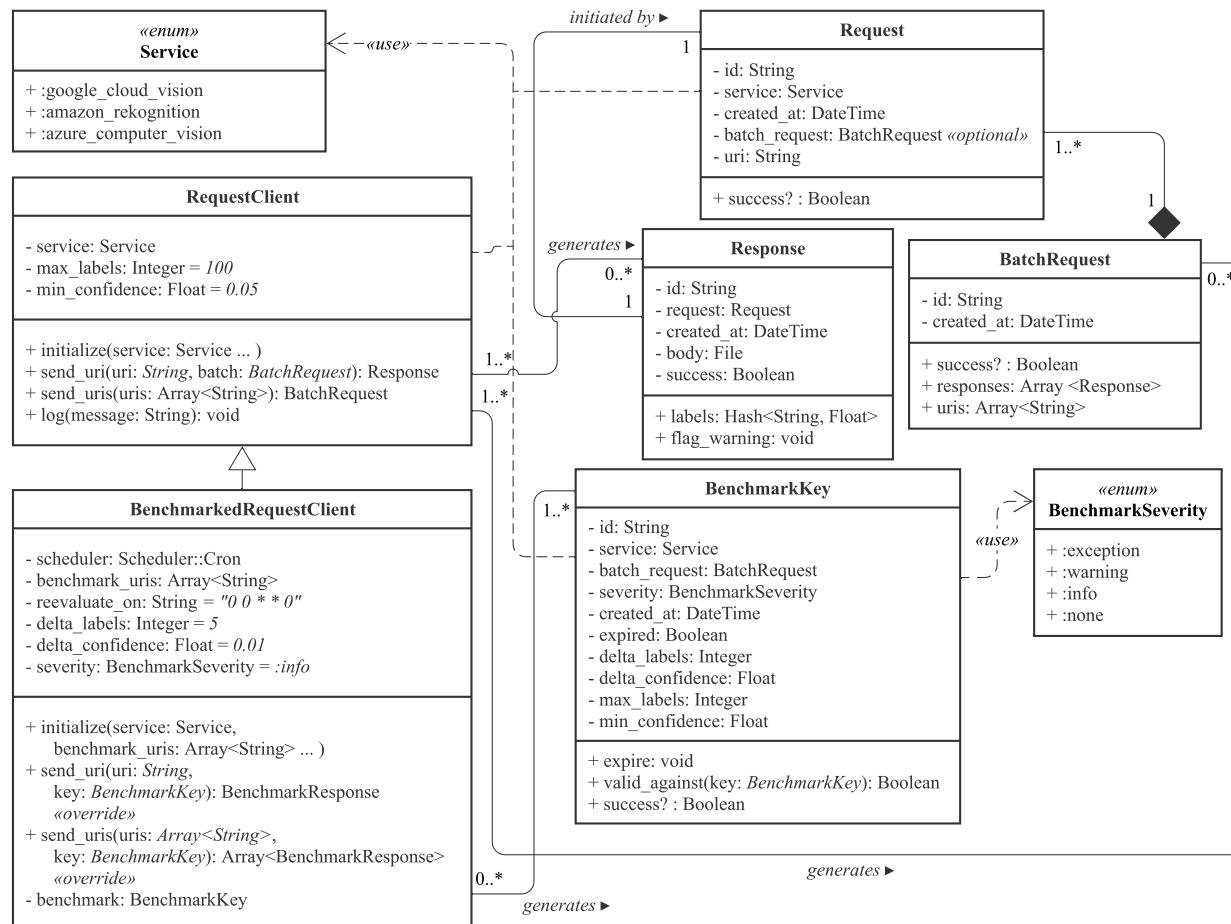


Figure A.10: Creation of a new benchmark proxy server using the architecture tactic.

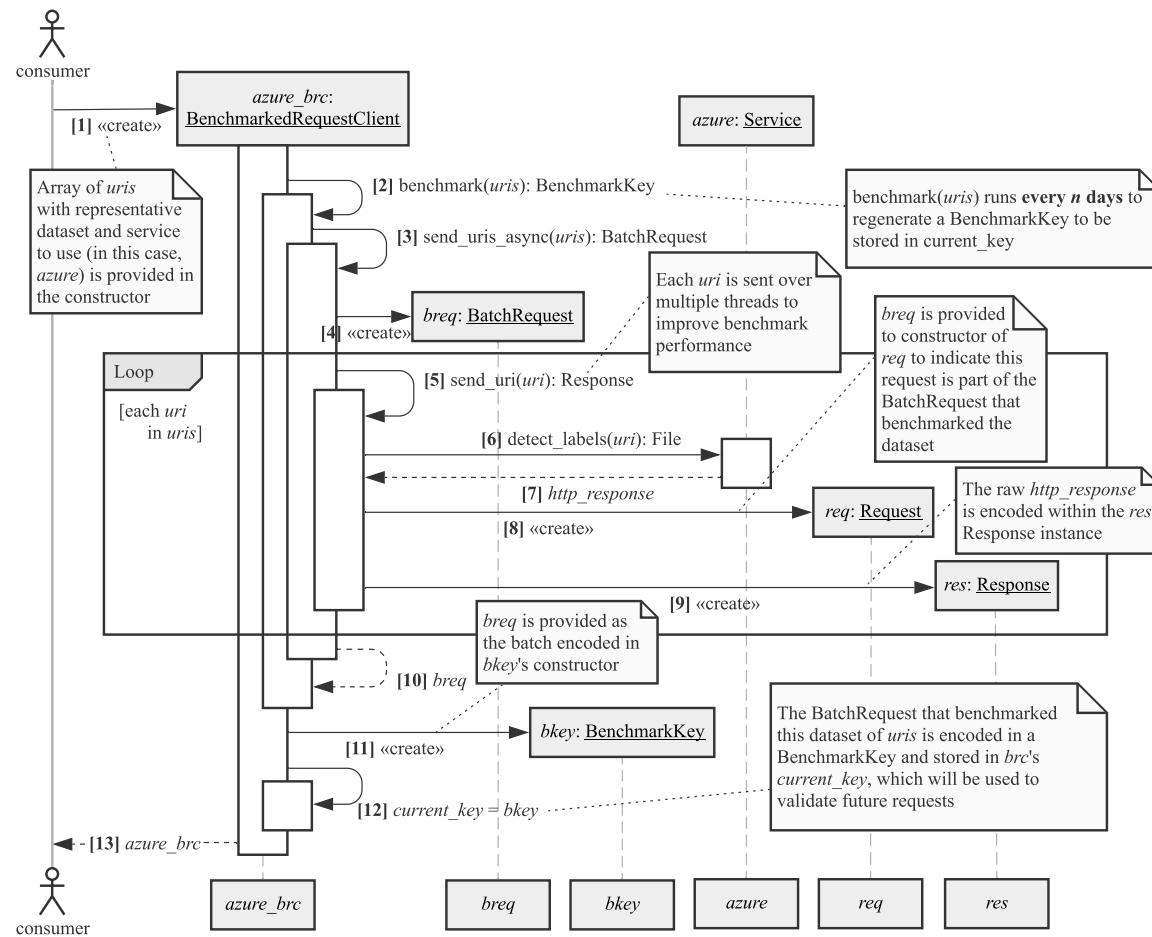


Figure A.11: Making a request through the proxy server ‘facade’.

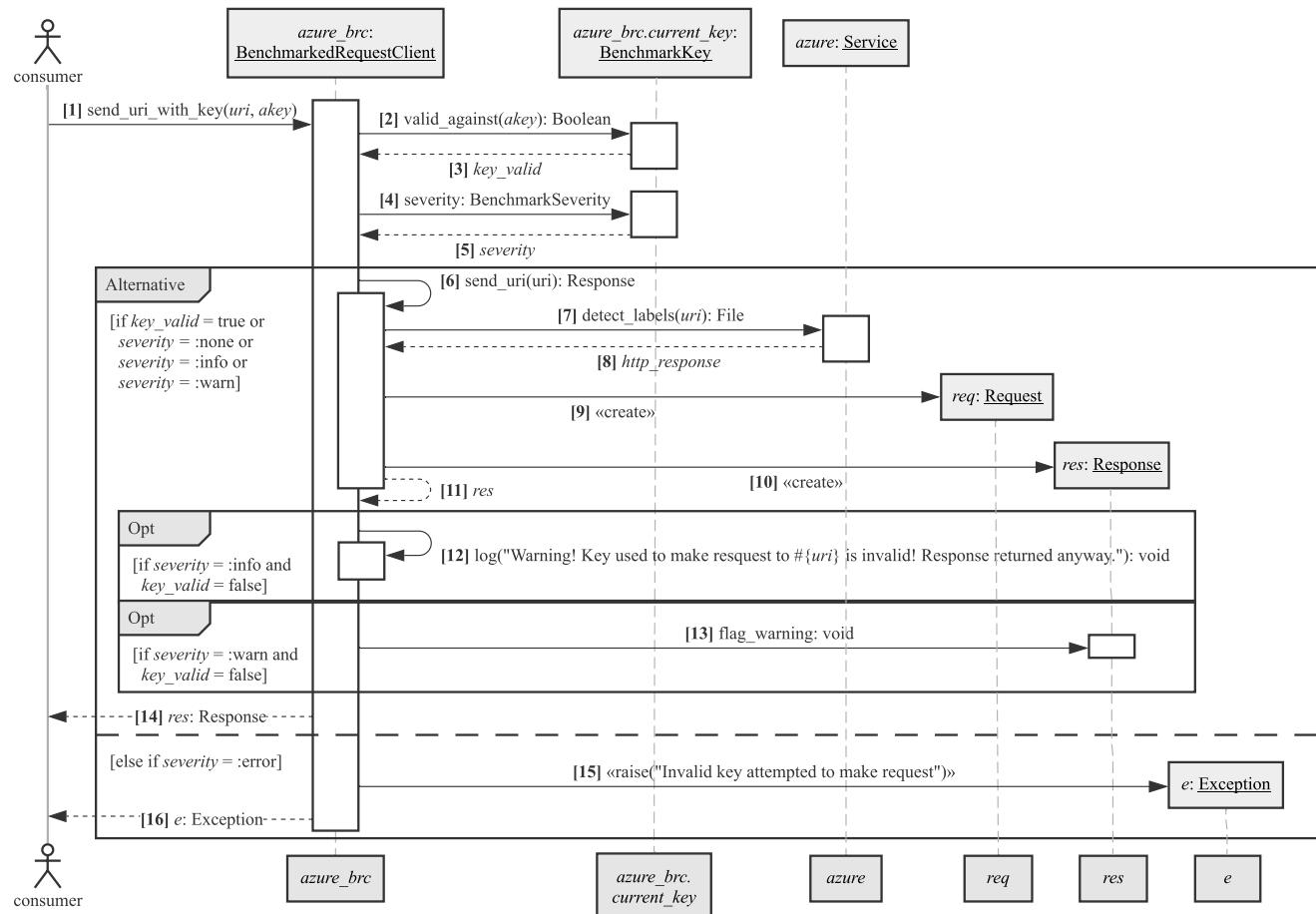


Figure A.12: State diagram of high-level workflows in the architectural tactic.

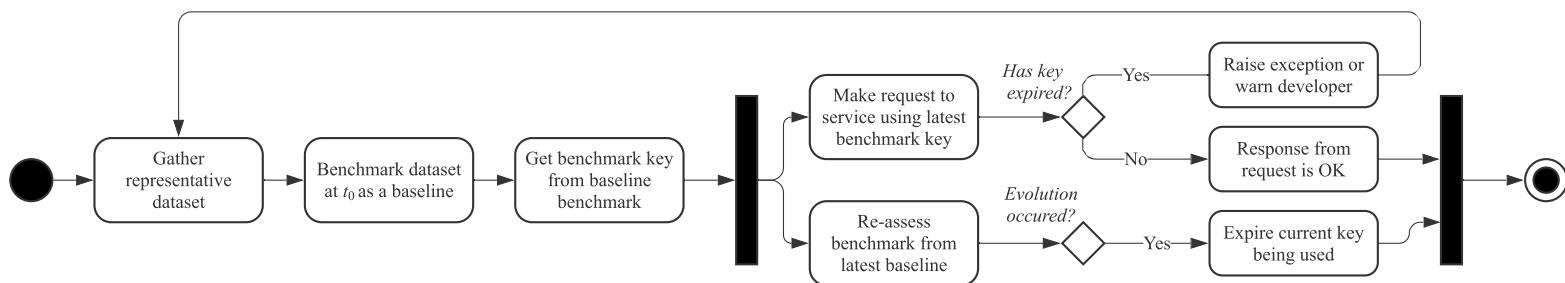
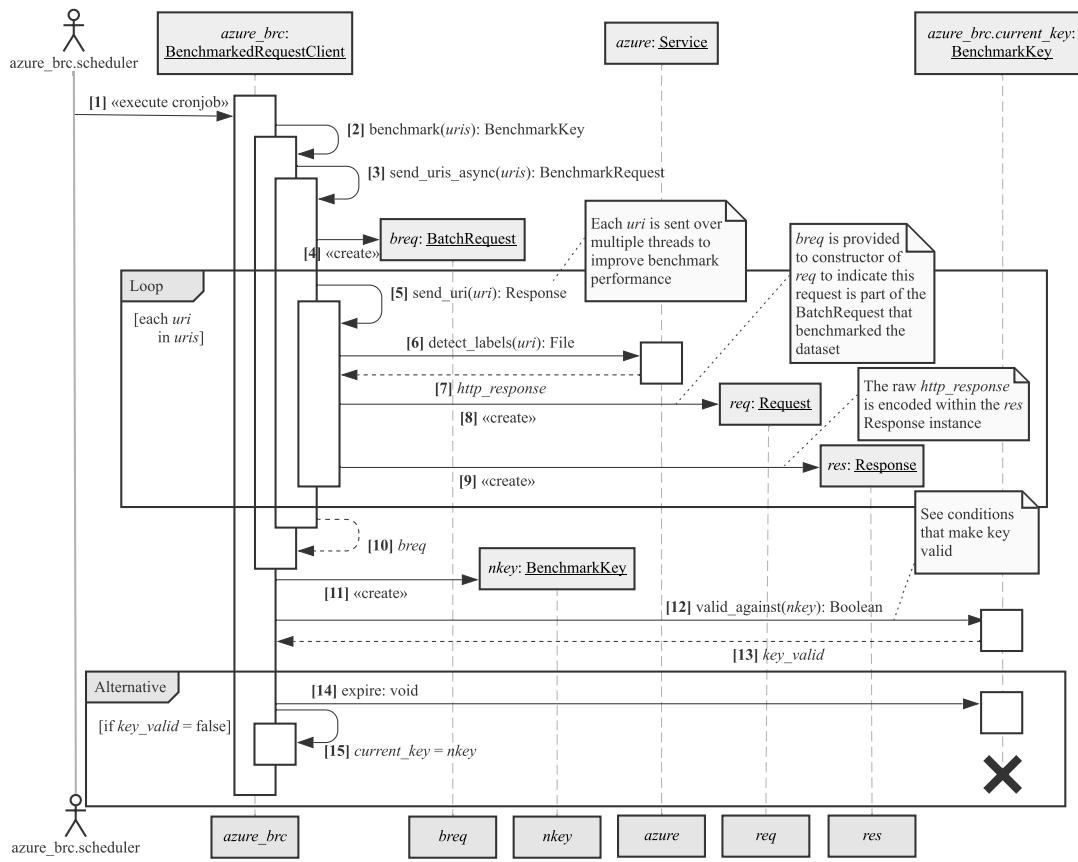


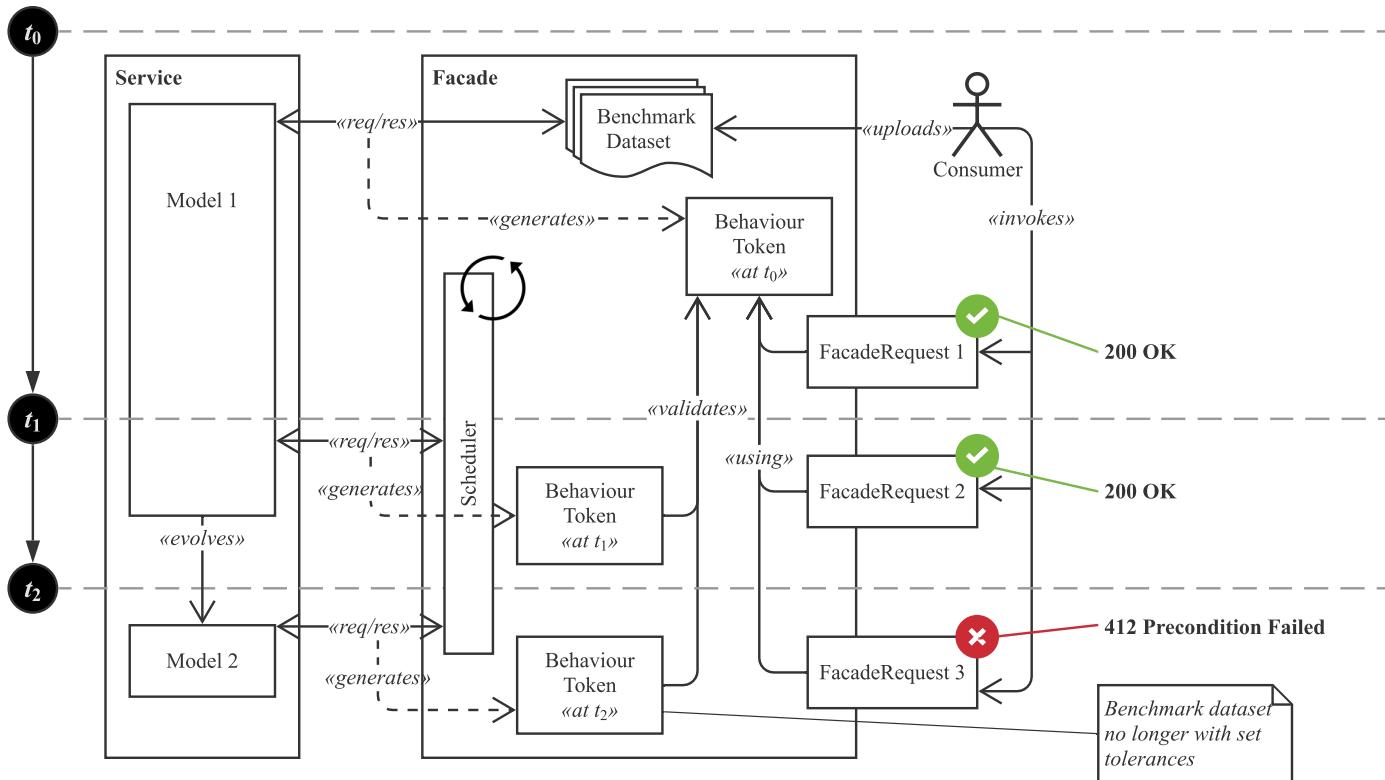
Figure A.13: Evolution occurring in the benchmark and how the architectural tactic notifies the consumer.



Conditions for a key to be valid

- both keys use the same services
- both keys encode the same URIs
- both keys have successful BatchRequests
- both keys must have BatchRequests with the same number of Response objects
- both keys must have the same cardinality of labels, within a margin of error of x
- for every label, each label must have a confidence value between both within a margin of error of y , i.e.: $\text{abs}(\text{conf}(\text{label}_i, \text{azure_brc.current_key}) - \text{conf}(\text{label}_i, \text{nkey})) \leq y$

Figure A.14: 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.

```



```

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       )

```

```

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   end
657   invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
658     BenchmarkKey::InvalidKeyError::LABEL_DELTA_MISMATCH, {
659       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   ICSVSB.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     ICSVSB.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 ICSVSB.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   ICSVSB.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 end
755
756 # Validates a response against this key as per rules encoded within this key.
757 # @param [Response] key The response to validate.
758 # @return See #valid_against?
759 def _validate_against_response(response)
760   invalid_key_errors = []
761
762   missing_expected_labels = expected_labels_set - Set[*response.labels.keys]
763   unless missing_expected_labels.empty?
764     invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
765       BenchmarkKey::InvalidKeyError::EXPECTED_LABELS_MISMATCH, {
766         source_key: {
767           id: id,
768           created_at: created_at
769         },
770         violating_response: {
771           id: response.id,
772           created_at: response.created_at,
773           body: response.hash
774         }
775       }
776     )
777   end
778 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}'. \"\n
780     \"However, response (id=#{response.id}) has the following labels: '#{\n
781       ↪ response.labels.keys.join(',')}'. \"\n
782     \"The following labels are missing: '#{missing_expected_labels.to_a.join
783       ↪ (',')}'.\"\n
784   }
785   )
786 end
787 end
788
789 # The Request Client class is used to make non-benchmarked requests to the
790 # provided service's labelling endpoints. It handles creating respective
791 # +Request+ and +Response+ records to be committed to the benchmarker database.
792 # Requests made with the +RequestClient+ do *not* ensure that evolution risk
793 # has occurred (see BenchmarkRequestClient).
794 class RequestClient
795   # Initialises a new instance of the requester to label endpoints.
796   # @param [Service] service The service to request from.
797   # @param [Fixnum] max_labels The maximum labels that the requester returns.
798   # Only supported if the service supports this parameter. Default is 100
799   # labels.
800   # @param [Float] min_confidence The confidence threshold by which labels
801   # are returned. Only supported if the service supports this parameter.
802   # Default is 0.50.
803   def initialize(service, max_labels: 100, min_confidence: 0.50)
804     unless service.is_a?(Service) && [Service::GOOGLE, Service::AMAZON, Service
805       ↪ ::AZURE, Service::DEMO].include?(service)
806       raise ArgumentError, "Service with name #{service.name} not supported."
807     end
808
809   # Registers logging for this client
810   ICVSB.register_request_client(self)
811   @logstrio = StringIO.new
812   @log = Logger.new(@logstrio)
813
814   @service = service
815   @service_client =
816     case @service
817     when Service::GOOGLE
818       Google::Cloud::Vision::ImageAnnotator.new
819     when Service::AMAZON
820       Aws::Rekognition::Client.new
821     when Service::AZURE
822       URI('https://australiaeast.api.cognitive.microsoft.com/vision/v2.0/tag')
823     when Service::DEMO
824       nil # Not client needed for mock...
825     end
826   @config = {
827     max_labels: max_labels,
828     min_confidence: min_confidence
829   }
830   @max_labels = max_labels
831   @min_confidence = min_confidence
832 end

```

```

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   end

```

```

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].include?(severity)
936       raise ArgumentError, 'Severity must be a Logger::Severity type'
937     end
938     raise ArgumentError, 'Message must be a string' unless message.is_a?(String)
939
940     @log.add(severity, message)
941   end
942
943   # Gets the log of this client as a string.
944   # @return [String] The entire log.
945   def read_log
946     @logstrio.string
947   end
948
949   # Makes a request to Google Cloud Vision's +LABEL_DETECTION+ feature.
950   # @see https://cloud.google.com/vision/docs/labels
951   # @param [String] uri A URI to an image to detect labels. Google Cloud
952   # 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
1257      # generated.

```

```

1257     def send_uri_with_key(uri, key)
1258       raise ArgumentError, 'URI must be a string.' unless uri.is_a?(String)
1259       raise ArgumentError, 'Key must be a BenchmarkKey.' unless key.is_a?(
1260         BenchmarkKey)
1261 
1262       if @current_key.nil?
1263         return {
1264           key_errors: [
1265             BenchmarkKey::InvalidKeyError.new(BenchmarkKey::InvalidKeyError::
1266               NO_KEY_YET)
1267           ]
1268         }
1269 
1270       result = {
1271         labels: nil,
1272         response: nil,
1273         key_errors: nil,
1274         response_errors: nil,
1275         service_error: nil,
1276         cached: nil
1277       }
1278 
1279       # Check for a cached result w/ this service given provided key...
1280       ICVSB.ldebug("Attempting to use a cached response for #{uri} + #{@service.
1281         name}...")
1282       Request.where(uri: uri, service_id: @service.id, benchmark_key_id: key.id)
1283         .order(Sequel.desc(:created_at)).each do |request|
1284         response = request.response
1285 
1286         # Ignore unsuccessful responses
1287         next if response.nil? || !response.success?
1288 
1289         # Check if the response's benchmark is still valid -- if so, just
1290         # reuse that result... (no need to actually ping service)
1291         key_is_valid, = @current_key.valid_against?(response.benchmark_key)
1292         ICVSB.ldebug("Cached key (id=#{response.benchmark_key.id}) is valid
1293           ↪ against current key \"\n
1294             "(id=#{@current_key.id})? #{key_is_valid}\")")
1295         if !response.benchmark_key.nil? && key_is_valid
1296           return { labels: response.labels, response: response.hash, cached:
1297             DateTime.parse(response.created_at.to_s) }
1298         end
1299       end
1300       ICVSB.ldebug("Cached response failed! Will try to invoke a request to #{@
1301         service.name}")
1302 
1303       # Check for key validity
1304       ICVSB.ldebug("Checking if current key (id=#{@current_key.id}) is valid
1305           ↪ against key provided (id=#{key.id})...")
1306       key_valid, key_invalid_reasons = @current_key.valid_against?(key)
1307       # Invalid state count incrementemnt if key error exists...
1308       unless key_valid
1309         ICVSB.ldebug("Validation of current key (id=#{@current_key.id}) failed
1310           ↪ against key provided (id=#{key.id}). "
1311             "Reasons: #[key_invalid_reasons.join('; ')]")
1312         result[:key_errors] = key_invalid_reasons
1313         @invalid_state_count += 1
1314         ICVSB.linfo("Error has occured in key validation. Invalid state count
1315           ↪ count is now #{@invalid_state_count}.")
1316       end
1317 
1318       # If key is valid, raise request and check if response is valid
1319       ICVSB.ldebug("Key provided #{key.id} is valid against current key #{
1320

```

```

1312     ↪ @current_key.id}!")
1313   if key_valid
1314     ICVSB.ldebug("Invoking a request '#{uri}' to #{@service.name}...")
1315     response = send_uri_no_key(uri)
1316     ICVSB.ldebug("Response returned (id=#{response.id})! Labels: #{response.
1317       ↪ labels}")
1318     # Update the benchmark key id
1319     response.benchmark_key_id = @current_key.id
1320     ICVSB.ldebug("Updated response (id=#{response.id}) with benchmark key = #{
1321       ↪ response.benchmark_key_id}...")
1322     # Now check to see if it was valid given that the response was successful
1323     if response.success?
1324       ICVSB.ldebug("Checking if this response (id=#{response.id}) is valid
1325         ↪ against current key (id=#{key.id})")
1326       response_valid, response_invalid_reasons = @current_key.valid_against?(
1327         ↪ response)
1328     end
1329     result[:labels] = response.labels
1330     result[:response] = response.hash
1331     result[:service_error] = result[:response][:service_error].to_s unless
1332       ↪ result[:response][:service_error].nil?
1333     response_valid ||= !result[:response][:service_error].nil?
1334     # Increment invalid state count if response error ONLY (i.e., not service
1335       ↪ error)
1336     unless response_valid
1337       ICVSB.ldebug("Validation of current key (id=#{@current_key.id}) failed
1338         ↪ against response \"\
1339           "(id=#{response.id}). Reasons: #{response_invalid_reasons.join('; ')}")
1340       result[:response_errors] = response_invalid_reasons
1341       @invalid_state_count += 1
1342       ICVSB.linfo('Error has occurred in response validation. \
1343           "Invalid state count count is now #{@invalid_state_count}.')
1344     end
1345   end
1346
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   #     break
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[:severity] != BenchmarkSeverity::WARNING
1451
1452      uri = @benchmark_config[:warning_callback_uri]
1453      data = result
1454      Thread.new do
1455          ICSVSB.linfo("POSTing to warning endpoint '#{uri}' data=#{data}")
1456          req = Net::HTTP::Post.new(uri)
1457          req.body = data.to_json
1458          req.content_type = 'application/json; charset=utf8'
1459          res = Net::HTTP.start(uri.hostname, uri.port) do |http|
1460              http.request(req)
1461          end
1462          ICSVSB.linfo("Response from warning endpoint: #{res.code} #{res.message}")
1463          ICSVSB.ldebug("Response body is: #{res.body}") if res.is_a?(Net::HTTPSuccess)
1464      end
1465  end
1466
1467  # Forwards a new key that has been generated due to benchmark trigger and
1468  # sends the current or old key (depending on expiry_occurred flag.)
1469  # @param [BenchmarkKey] new_key The new key that was generated from the
1470  # benchmark that was triggered.
1471  # @param [BenchmarkKey] old_or_current_key The current key, if expiry did
1472  # not occur, or the old key if expiry did occur.
1473  # @param [Boolean] expiry_occurred Indicates if the current_key was expired
1474  # and replaced with the new_key.
1475  # @return [void]
1476  def _flag_benchmarking_complete(new_key, old_or_current_key, expiry_occurred)
1477      return if @benchmark_config[:benchmark_callback_uri].nil?
1478
1479      uri = @benchmark_config[:benchmark_callback_uri]
1480      old_or_current_key_id = old_or_current_key.nil? ? nil : old_or_current_key.id
1481      data = { new_key: new_key.id, old_key: old_or_current_key_id, expiry_occurred: expiry_occurred }
1482      Thread.new do
1483          ICSVSB.linfo("POSTing to benchmark complete endpoint '#{uri}' data=#{data}")
1484          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 # configurated 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 |
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 |
459 |   send_file(File.join(settings.demo_folder, image_id + '.' + ext))
```

APPENDIX C

Supplementary Materials to Chapter 8

C.1 Detailed Overview of Our Proposed Taxonomy

The following pages detail our proposed taxonomy. Detailed descriptions of the five requirements of good API documentation (dimensions) and 34 generalised API documentation artefacts (categories/sub-dimensions) that help satisfy these requirements within our proposed taxonomy. Descriptions of examples of these documentation artefacts are italicised and provided for illustrative purposes. ILS = In-Literature Score, calculated as a ratio of papers that investigated or reported various issues concerning each artefact. IPS = In-Practice Score, calculated as the average response from our survey instrument. 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
A Requirement 1: API Documentation should include Descriptions of API Usage							
A1	Quick-start guides; <i>i.e., a guide to rapidly get started using the API in a specific programming language.</i>	S4, S9, S10	Low	V High	●	○	●
A2	Low-level reference manual; <i>i.e., a manual documenting all API components to review fine-grade detail.</i>	S1, S3, S4, S8, S9, S10, S11, S12, S15, S16, S17	High	High	●	●	●
A3	Explanation of high level architecture; <i>i.e., explanations of the API's high-level architecture to better understand intent and context.</i>	S1, S2, S4, S11, S14, S16, S19, S20	Med	V High	●	●	●
A4	Introspection source code comments; <i>i.e., code implementation and code comments (where applicable) to understand the API author's mindset.</i>	S1, S4, S7, S12, S13, S17, S20	Med	High	○	○	○
A5	Code snippets of basic component function; <i>i.e., code snippets (with comments) of no more than 30 LoC to understand a basic component functionality within the API.</i>	S1, S2, S4, S5, S6, S7, S9, S10, S11, S14, S15, S16, S18, S20, S21	V High	V High	●	●	●
A6	Step-by-step tutorials with multiple components; <i>i.e., step-by-step tutorials, with screenshots to understand how to build a non-trivial piece of functionality with multiple components of the API.</i>	S1, S2, S4, S5, S7, S9, S10, S15, S16, S18, S20, S21	V High	V High	○	●	●
A7	Downloadable production-ready source code; <i>i.e., downloadable source code of production-ready applications that use the API to understand implementation in a large-scale solution.</i>	S1, S2, S5, S9, S15	Low	V High	○	○	●
A8	Best-practices of implementation; <i>i.e., best-practices of implementation to assist with debugging and efficient use of the API.</i>	S1, S2, S4, S5, S7, S8, S9, S14	Med	V High	○	●	○
A9	An exhaustive list of all components; <i>i.e., a list of all the major components that exist within the API.</i>	S4, S16, S19	Low	V High	○	●	●
A10	Minimum system requirements to use the API; <i>i.e., requirements and the dependencies to use the API on a particular system.</i>	S4, S7, S13, S17, S19	Low	V High	●	○	○
A11	Instructions to install/update the API and its release cycle; <i>i.e., instructions to install or begin using the API and details on its release cycle and how to update it.</i>	S4, S7, S8, S9, S11, S13, S16, S19	Med	V High	●	●	○
A12	Error definitions describing how to address problems	S1, S2, S4, S5, S9, S11, S13	Med	V High	○	○	○

Continued on next page...

Key	Description	Primary Sources	ILS	IPS	GCV	AWS	ACV
B Requirement 2: API Documentation should include Descriptions of the API's Design Rationale							
B1	Entry-point purpose of the API; <i>i.e., a brief description of the purpose or overview of the API as a low barrier to entry.</i>	S1, S2, S4, S5, S6, S8, S10, S11, S15, S16	High	V High	●	●	●
B2	What the API can develop; <i>i.e., descriptions of concrete types of applications the API can develop.</i>	S2, S4, S9, S11, S15, S18	Med	V High	●	●	●
B3	Who should use the API; <i>i.e., descriptions of the types of users who should use the API.</i>	S4, S9	V Low	High	●	○	○
B4	Who will use the applications built using the API; <i>i.e., descriptions of the types of users who will use the product the API creates.</i>	S4	V Low	Med	○	○	○
B5	Success stories on the API; <i>i.e., example success stories of major users that describe how well the API was used in production.</i>	S4	V Low	V High	●	●	●
B6	Documentation comparing similar APIs to this API	S2, S6, S13, S18	Low	High	●	○	●
B7	Limitations on what the API can/cannot provide	S4, S5, S8, S9, S14, S16	Med	V High	○	●	●
C Requirement 3: API Documentation should include Descriptions of the Domain Concepts behind the API							
C1	Relationship between API components and domain concepts	S3, S10	V Low	High	○	○	●
C2	Definitions of domain terminology; <i>i.e., definitions of the domain-terminology and concepts, with synonyms if applicable.</i>	S2, S3, S4, S6, S7, S10, S14, S16	Med	V High	●	○	●
C3	Documentation for nontechnical audiences; <i>i.e., generalised documentation for non-technical audiences regarding the API and its domain.</i>	S4, S8, S16	Low	High	●	●	●
D Requirement 4: API Documentation should include Additional Support Artefacts to aide Developer Productivity							
D1	FAQs	S4, S7	V Low	V High	●	●	●
D2	Troubleshooting hints	S4, S8	V Low	High	○	●	○○
D3	API diagrams; <i>i.e., diagrammatically representing API components using visual architectural representations.</i>	S6, S13, S20	Low	V High	○	○	○○
D4	Contact for technical support	S4, S8, S19	Low	Med	●	●	●
D5	Printed guide	S4, S6, S7, S9, S16	Low	V High	○	●	●

Continued on next page...

Key	Description	Primary Sources	ILS	IPS	GCV	AWS	ACV
D6	Licensing information	S7	V Low	V High	○	○	●
E Requirement 5: API Documentation should be Presented in an Easily Digestible Format							
E1	Searchable knowledge base	S3, S4, S6, S10, S14, S17, S18	Med	V High	●	●	●
E2	Context-specific discussion forums	S4, S10, S11	Low	V High	●	●	●
E3	Quick-links to other relevant components	S6, S16, S20	Low	V High	○	○	○
E4	Structured navigation style; <i>i.e.</i> , <i>breadcrumbs</i>	S6, S10, S20	Low	High	●	●	●
E5	Visualised map of navigational paths; <i>i.e.</i> , <i>to certain API components in the website</i> .	S6, S14, S20	Low	V High	○	○	○
E6	Consistent look and feel	S1, S2, S3, S5, S6, S8, S10, S15, S20	High	V High	●	●	●

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/#insight-from-your-images https://developers.google.com/machine-learning/glossary/ https://cloud.google.com/vision/docs/resources https://cloud.google.com/vision/sla https://cloud.google.com/vision/docs/data-usage https://cloud.google.com/vision/docs/support#searchbox https://cloud.google.com/vision/docs/support

Continued on next page...

Service	Document Sources
Amazon Rekognition	<p>https://docs.aws.amazon.com/rekognition/latest/dg/getting-started.html</p> <p>https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/index.html</p> <p>https://aws.amazon.com/blogs/machine-learning/using-amazon-rekognition-to-identify-persons-of-interest-for-law-enforcement/</p> <p>https://aws.amazon.com/rekognition/#Rekognition_Image_Use_Cases</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/labels-detect-labels-image.html</p> <p>https://aws.amazon.com/rekognition/getting-started/#Tutorials</p> <p>https://aws.amazon.com/blogs/machine-learning/category/artificial-intelligence/amazon-rekognition/</p> <p>https://docs.aws.amazon.com/code-samples/latest/catalog/code-catalog-javascript-example_code-rekognition.html</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/best-practices.html</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/API_Operations.html</p> <p>https://aws.amazon.com/rekognition/image-features/</p> <p>https://aws.amazon.com/releasenotes/?tag=releasenotes%23keywords%23amazon-rekognition</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/setting-up.html</p> <p>https://aws.amazon.com/rekognition/</p> <p>https://aws.amazon.com/rekognition/</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/limits.html</p> <p>https://aws.amazon.com/rekognition/pricing/</p> <p>https://aws.amazon.com/rekognition/sla/</p> <p>https://aws.amazon.com/rekognition/faqs/</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/video-troubleshooting.html</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/rekognition-dg.pdf</p> <p>https://github.com/awsdocs/amazon-rekognition-developer-guide/issues</p> <p>https://forums.aws.amazon.com/thread.jspa?threadID=285910</p>

Continued on next page...

Service	Document Sources
Azure Computer Vision	https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/quickstarts-sdk/csharp-analyze-sdk https://docs.microsoft.com/en-us/java/api/overview/azure/cognitiveservices/client/computervision?view=azure-java-stable https://docs.microsoft.com/en-us/azure/architecture/example-scenario/ai/intelligent-apps-image-processing https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/tutorials/java-tutorial https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/logo-detector-mobile https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/tutorials/storage-lab-tutorial https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/tutorials/csharptutorial https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/getting-started-improving-your-classifier https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/home#analyze-images-for-insight https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/vision-api-how-to-topics/howtocallvisionapi https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/release-notes https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/ https://azure.microsoft.com/en-au/services/cognitive-services/computer-vision/ https://azure.microsoft.com/en-us/pricing/details/cognitive-services/computer-vision/ https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/concept-tagging-images https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/home https://azure.microsoft.com/en-us/support/legal/sla/cognitive-services/v1_1/ https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/faq https://azure.microsoft.com/en-us/support/legal/

C.3 List of Primary Sources

The following pages list of the primary sources found from our systematic mapping study. Each citation is referenced by a prefixed ‘S’. We also list the respective citation count, as measured by the number of citations the publication has from Google Scholar as at July 2020. We also list the venue ranking (as at 2020), as measured by Scimago Rankings or Qualis Ranking for Journals and CORE Rankings for conference publications. If no rank can be found, a dash is used.

Ref	Citation	Cite#	Rank
[S1]	M. P. Robillard, "What makes APIs hard to learn? Answers from developers," <i>IEEE Software</i> , vol. 26, no. 6, pp. 27–34, 2009, DOI 10.1109/MS.2009.193. ISSN 0740-7459	305	Q1
[S2]	M. P. Robillard and R. Deline, "A field study of API learning obstacles," <i>Empirical Software Engineering</i> , vol. 16, no. 6, pp. 703–732, 2011, DOI 10.1007/s10664-010-9150-8. ISSN 1382-3256	254	Q1
[S3]	A. J. Ko and Y. Riche, "The role of conceptual knowledge in API usability," in <i>Proceedings of the 2011 IEEE Symposium on Visual Languages and Human Centric Computing</i> . Pittsburgh, PA, USA: IEEE, September 2011. DOI 10.1109/VL-HCC.2011.6070395. ISBN 978-1-45-771245-6 pp. 173–176	33	A
[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 <i>Proceedings of the 20th Annual International Conference on Computer Documentation</i> . Toronto, ON, Canada: ACM, October 2002. DOI 10.1145/584955.584976, pp. 133–141	56	–
[S5]	R. Watson, M. Mark Stammes, J. Jeannot-Schroeder, and J. H. Spyridakis, "API documentation and software community values: A survey of open-source API documentation," in <i>Proceedings of the 31st ACM International Conference on Design of Communication</i> . Greenville, SC, USA: ACM, September 2013. DOI 10.1145/2507065.2507076, pp. 165–174	14	B1
[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 <i>Proceedings of the First International Symposium on End User Development</i> , vol. 5435 LNCS. Siegen, Germany: Springer, March 2009. DOI 10.1007/978-3-642-00427-8_6. ISSN 0302-9743 pp. 86–105	34	–
[S7]	E. Aghajani, C. Nagy, O. L. Vega-Marquez, M. Linares-Vasquez, L. Moreno, G. Bavota, and M. Lanza, "Software Documentation Issues Unveiled," in <i>Proceedings of the 41st International Conference on Software Engineering</i> . Montreal, QC, Canada: IEEE, May 2019. DOI 10.1109/ICSE.2019.00122. ISBN 978-1-72-810869-8. ISSN 0270-5257 pp. 1199–1210	6	A*
[S8]	S. Haselbock, R. Weinreich, G. Buchgeher, and T. Kriechbaum, "Microservice Design Space Analysis and Decision Documentation: A Case Study on API Management," in <i>Proceedings of the 11th International Conference on Service-Oriented Computing and Applications</i> , Paris, France, November 2019, DOI 10.1109/SOCA.2018.00008, pp. 1–8	2	C
[S9]	S. Inzunza, R. Juárez-Ramírez, and S. Jiménez, "API Documentation," in <i>Proceedings of the 6th World Conference on Information Systems and Technologies</i> . Naples, Italy: Springer, March 2018. DOI 10.1007/978-3-319-77712-2_22, pp. 229–239	3	C

Continued on next page...

Ref	Citation	Cite#	Rank
[S10]	M. Meng, S. Steinhardt, and A. Schubert, "Application programming interface documentation: What do software developers want?" <i>Journal of Technical Writing and Communication</i> , vol. 48, no. 3, pp. 295–330, August 2018, DOI 10.1177/0047281617721853. ISSN 1541-3780	12	Q1
[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," <i>Computer Supported Cooperative Work: CSCW: An International Journal</i> , vol. 27, no. 3-6, pp. 767–802, May 2018, DOI 10.1007/s10606-018-9333-1. ISSN 1573-7551	4	Q1
[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 <i>Proceedings of the 40th International Conference on Software Engineering</i> , 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	4	A*
[S13]	L. Aversano, D. Guardabascio, and M. Tortorella, "Analysis of the Documentation of ERP Software Projects," <i>Procedia Computer Science</i> , vol. 121, pp. 423–430, January 2017, DOI 10.1016/j.procs.2017.11.057. ISSN 1877-0509	4	–
[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 <i>Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution</i> . Shanghai, China: IEEE, September 2017. DOI 10.1109/ICSME.2017.17, pp. 479–483	55	A*
[S15]	R. Watson, "Development and application of a heuristic to assess trends in API documentation," in <i>Proceedings of the 30th ACM International Conference on Design of Communication</i> . Seattle, WA, USA: ACM, October 2012. DOI 10.1145/2379057.2379112. ISBN 978-1-45-031497-8 pp. 295–302	10	B1
[S16]	W. Maalej and M. P. Robillard, "Patterns of knowledge in API reference documentation," <i>IEEE Transactions on Software Engineering</i> , 2013, DOI 10.1109/TSE.2013.12. ISSN 0098-5589	110	Q1
[S17]	D. L. Parnas and S. A. Vilkomir, "Precise documentation of critical software," in <i>Proceedings of 10th IEEE International Symposium on High Assurance Systems Engineering</i> . Plano, TX, USA: IEEE, November 2007. DOI 10.1109/HASE.2007.63. ISSN 1530-2059 pp. 237–244	2	B
[S18]	C. Bottomley, "What part writer? What part programmer? A survey of practices and knowledge used in programmer writing," in <i>Proceedings of the 2005 IEEE International Professional Communication Conference</i> . Limerick, Ireland: IEEE, July 2005. DOI 10.1109/IPCC.2005.1494255, pp. 802–812	0	–

Continued on next page...

Ref	Citation	Cite#	Rank
[S19]	A. Taulavuori, E. Niemelä, and P. Kallio, “Component documentation - A key issue in software product lines,” <i>Information and Software Technology</i> , vol. 46, no. 8, pp. 535–546, June 2004, DOI 10.1016/j.infsof.2003.10.004. ISSN 0950-5849	40	Q1
[S20]	J. Kotula, “Using patterns to create component documentation,” <i>IEEE Software</i> , vol. 15, no. 2, pp. 84–92, 1998, DOI 10.1109/52.663791. ISSN 0740-7459	27	Q1
[S21]	S. G. McLellan, A. W. Roesler, J. T. Tempest, and C. I. Spinuzzi, “Building more usable APIs,” <i>IEEE Software</i> , vol. 15, no. 3, pp. 78–86, 1998, DOI 10.1109/52.676963. ISSN 0740-7459	105	Q1

C.4 Detailed Suggested Improvements

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

C.4.1 Issues regarding Descriptions of API Usage

Quick-start guides [A1]: Quick-start guides should provide a short tutorial that allows programmers to pick up the basics of an API in a programming language of their choice. For the services assessed, each offer various client SDKs (e.g., as Java or Python client libraries). Google Cloud Vision and Azure Computer Vision offer quick-start guides [425, 444] in which sets of articles target various SDKs or are client-agnostic with code snippets that can be changed to the client language/SDK of the developer's choice. Amazon Rekognition offers exercises in setting up the AWS SDK and using the command-line interface to interact with image analysis components [403], however this is client-agnostic nor does it provide details in how to get started with using the client SDKs.

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

Step-by-step tutorials [A6]: Google Cloud Vision offers tutorials limited to one component. These do not sufficiently demonstrate how to combine *multiple components* of the API together and how developers should integrate it with a different platform, which a good step-by-step tutorial should detail. The official AWS Machine Learning blog [400] provides extensive tutorials (in some cases, with a suggested tutorial completion time of over an hour) that integrate multiple Amazon Rekognition components with other AWS components. Microsoft provide tutorials [441, 447, 448] integrating multiple components within their service to mobile 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.

Downloadable production-ready applications [A7]: Microsoft provide a downloadable application [446] that explores many components of the Azure Computer Vision API. The application is thoroughly documented with and also provides guidance on how to structure the architecture design of the program. While Rekognition

and Google Cloud Vision also provide downloadable source code, they are largely under-documented, do not combine multiple components of the API together, and only use god-classes to handle all requests to the API [404, 427].

 **Suggested improvement:** Downloadable source code should be thoroughly documented, 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.

Understanding best-practices [A8]: Google Cloud provides best-practices for its platform in both general and enterprise contexts [419, 428], but there is little advice provided to guide developers on how best to use Google Cloud Vision. Microsoft provides guidance on improving results of custom vision classifiers [442], but no further details on non-custom vision classifiers are found. We found the most detailed best-practices to be provided by Amazon Rekognition [402], which outlines more detailed strategies such as reducing data transfer by storing and referencing images on S3 Buckets or the attributes images should have in various scenarios (e.g., the angles of a person's face in facial recognition).

 **Suggested improvement:** Document best-practices for all major components of the computer vision service. 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.

Exhaustive lists of all major API components [A9]: Amazon provides a two-fold feature list that describes both the key features of Rekognition at a high-level [401] as well as a detailed, technical breakdown of each API operation provided within the service [399]. Microsoft also provide a list of high-level features that Azure Computer Vision can analyse [449] which provides hyperlinks to detailed descriptions of each feature. Google's Cloud Vision API provides a partial breakdown of the types of services provided, however this list is not fully complete, nor are there hyperlinks to more detailed descriptions of each of the features [429].

 **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.

Minimum system requirements and dependencies [A10]: Although there is no dedicated webpage for this on any of the services investigated, there are listed dependencies for the client libraries in Google's and Azure's quick-start guides [425, 439]. These may be embedded within the quick-start guide as developers are likely to encounter dependency issues when they first start using the API. We found it a 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.

Installation and release cycle notes [A11]: It is imperative that developers know what has changed between releases and how frequently the releases are exported. We found release notes for Amazon Computer Vision, although they are only major releases and have not been updated since 2017 [398] which does not account for evolution in the service's responses [88]. Google's and Microsoft's release notes are generally more frequently updated, therefore developers can get a sense of its release frequency [426, 445]. However, there are evolution issues that are not addressed. Installation instructions are detailed within Rekognition's developer guide, outlining how to sign up for an account, and install the AWS command-line interface [406].

☞ **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 computer vision service so that developers are aware of potential side-effects of upgrading to a newer release.

C.4.2 Issues regarding Descriptions of Design Rationale

Limitations of the API [B7]: The most detailed limitations documented were found on Rekognition's dedicated limitations page [405] that outlines functional limitations such as the maximum number of faces or words that can be detected in an image, the size requirements of images, and file type information. For the other services, functional limitations are generally found within each endpoint's API 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.

C.4.3 Issues regarding Descriptions of Domain Concepts

Conceptual understanding of the API [C1]: Azure Computer Vision provides 'concept' pages describing the high-level concepts behind computer vision and where these functions are implemented within the APIs (e.g., [440]). We were unable to 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.

Definitions of domain-specific terminology [C2]: Terminologies relevant to machine learning concepts powering these computer vision services are well detailed within Google’s machine learning glossary [423], however few examples matching computer vision are immediately relevant. While this page is linked from the original Google Cloud Vision documentation, it may be too technical for application developers to grasp. A slightly better example of this is [449], where developers can understand computer vision terms in lay terms.

↳ Suggested improvement: *Current computer vision services 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 computer vision services 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.*

C.4.4 Issues regarding Existence of Support Artefacts

Troubleshooting suggestions [D2]: The only troubleshooting tips found in our analysis were in Rekognition’s video service [407]. Further detailed instances of these troubleshooting tips could be expanded to non-video issues. For instance, if developers upload ‘noisy’ images, how can they inform the system of a specific ontology to use or to focus on parts of the foreground or background of the image? These are suggestions which we have proposed in prior work [88] that do not seem to be documented.

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

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

C.5 Survey Questions

This section contains the exact text of the survey described in Section 8.5.1. Our instrument also included questions where answers were not included in the research reported in this article, e.g. questions 1 and 2 regarding consent and ensuring participants have had development experience. Images used within the survey have been removed.

Developer opinions towards the importance of web API documentation recommendations

In this study, we are finding out how important recommendations of web API documentation are to developers. From this, we will improve AI-powered APIs. While there are screenshots of example APIs in the questions, think of an API that you have used based on **your own prior experience** when answering these questions. Thanks for taking the time to answer these questions; it should only take you about **10–20 minutes** to complete.

Attribution Notice

Portions of this questionnaire are reproduced from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.

Implementation-specific documentation of web APIs

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the statement. For each question, please nominate how much you agree with the following statements: *[Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree]*

- Q3a. I think quick-start guides with code that help me get started with an API's client library are important. e.g., quick-start guides that show how to get started and interact with the API and its responses.
- Q3b. I don't find low-level documentation of all classes and methods particularly helpful. e.g., a generated online reference manual from Javadoc comments.
- Q3c. I would imagine that explanations of the API's high-level architecture, context and rationale would be important to better understand how to consume the API. e.g., a graphic showing how the API could fit into the wider context of an application.
- Q3d. If I want to understand why an API did something that I didn't expect, the source code comments generally don't help me. e.g., an example from the Lodash API that describes why `set.add` isn't directly returned.
- Q3e. I find small code snippets with comments to demonstrate a single component's basic functionality within the API a useful way to learn. e.g., 10-30 lines of code to demonstrating various how-tos of a computer vision API.
- Q3f. I think it's cumbersome to read through step-by-step tutorials that show how to build something non-trivial with multiple components using the API. e.g., a ten-step tutorial documenting how to combine face recognition, face analysis, scene description, and landmark detection API components to generate descriptions of photos.

- Q3g. I think it's useful to download source code of production-ready applications that demonstrate the use of multiple facets of the API. e.g., a downloadable iOS app that demonstrates how to perform image analysis on an iPhone/iPad.
- Q3h. I think official documentation describing the 'best-practices' of how to use the API to assist with debugging and efficiency is not helpful. e.g., an article describing the correct ways of doing things, the best tools to use, and how to write well-performing code.
- Q3i. I believe an exhaustive list of all major components in the API without excessive detail would be useful when learning an API. e.g., a computer vision web API might list object detection, object localisation, facial recognition, and facial comparison as its 4 components.
- Q3j. I believe minimum system requirements and/or dependencies to use the API do not always need to be part of official documentation. e.g., I can find descriptions of how to get started with a Python environment for a cloud platform on community forums instead of the API's website.
- Q3k. I think instructions on how to install or access the API, update it, and the frequency of its release cycle is all useful information to know about. e.g., a list showing the latest releases, what was added and how to update your application to make use of it.
- Q3l. Error codes describing specific problems with an API are not helpful. e.g., a list of canonical HTTP error codes and how to interpret them.

Rationale-specific documentation of web APIs

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the statement. For each question, please nominate how much you agree with the following statements: [*Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree*]

- Q4a. I think that, as a starting point when beginning to learn about an API, I would like to read about descriptions of the API's purpose and overview.
- Q4b. I don't find descriptions of the types of applications the API can develop helpful.
- Q4c. I believe that descriptions of the types of developers who should and shouldn't use the API is important to know.
- Q4d. I don't think that descriptions of the types of end-users who will use the product built using the API is important to know in advance.
- Q4e. I think that if I read success stories about when the API was previously used in production, I would have a better indicator of how I could use that API.
- Q4f. I think that documentation that compares an API to other, similar APIs confusing and not important.
- Q4g. I believe it is important to know about what the limitations are on what the API can and cannot provide.

Conceptual-specific documentation of web APIs

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the

statement. For each question, please nominate how much you agree with the following statements: [*Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree*]

- Q5a. I wouldn't read through theory about the API's domain that relates theoretical concepts to API components and how both work together.
 - Q5b. I think it is important to know the definitions of the API's domain-specific terminology and concepts (with synonyms where needed). e.g., a computer vision API that uses machine learning should list machine learning concepts.
 - Q5c. It's not really important to document information about the API to non-technical audiences, such as managers and other stakeholders. e.g., pricing information, uptime information, QoS metrics/SLAs etc.
-

General-support documentation of web APIs

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the statement. For each question, please nominate how much you agree with the following statements: [*Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree*]

- Q6a. I find lists of Frequently Asked Questions (FAQs) helpful.
 - Q6b. When something goes wrong, I don't read through troubleshooting suggestions for specific problems straight away as I like to solve it myself.
 - Q6c. I like to see diagrammatic representations of an API's components using visual architectural visualisations. e.g., UML class diagram, sequence diagram.
 - Q6d. I wouldn't look for email addresses and/or phone number for technical support in an API's documentation.
 - Q6e. I generally refer to a programmer's reference guide or textbook about the API when I need to.
 - Q6f. I don't think it's important to read about the licensing information about the API.
-

The effect of structure and tooling on web API documentation

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the statement. For each question, please nominate how much you agree with the following statements: [*Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree*]

- Q7a. I would like to use a searchable knowledge base to find information.
- Q7b. I think a context-specific discussion forum between developers isn't very helpful as it just introduces noise. e.g., issue trackers, Slack group.
- Q7c. I think links to other similar documentation frequently viewed by other developers would be useful. e.g., 'people who viewed this also viewed...'
- Q7d. If I get lost within the API's documentation, a 'breadcrumbs'-style of navigation isn't very useful to me.

- Q7e. A visualised map of navigational paths to common API components in the website would be useful to have. e.g., a large and complex API for Enterprise Service-Oriented Architecture where I could click into various boxes to read about components and arrows to read about how they are related.
- Q7f. I believe ensuring consistent look and feel of all documentation isn't necessary to a good API documentation.
-

Demographics

- Q8a. Are you, or do you aspire to be, a professional programmer? Or would you consider programming a hobby?
[Professional, Hobbyist]
- Q8b. How many years have you been programming?
[1–5 years, 6–10 years, 11–15 years, 16–20 years, 21–30 years, 31–40 years, 41+ years]
- Q8c. In what type of role would you say your current job falls into?
[Back-end developer, Data or business analyst, Data scientist or machine learning specialist, Database administrator, Designer, Desktop or enterprise applications developer, DevOps specialist, Educator or academic researcher, Embedded applications or devices developer, Engineering manager, Front-end developer, Full-stack developer, Game or graphics developer, Marketing or sales professional, Mobile developer, Product manager, QA or test developer, Student, System administration]
- Q8d. What level of seniority would you say this role falls into?
[Intern Role, Graduate Role, Junior Role, Mid-Tier Role, Senior Role, Lead Role, Principal Role, Management, N/A (e.g., I am a student), Other]
- Q8e. What industry would you say you work in?
[Cloud-based solutions or services, Consulting, Data and analytics, Financial technology or services, Healthcare technology or services, Information technology, Media, advertising, publishing, or entertainment, Other software development, Retail or eCommerce, Software as a service (SaaS) development, Web development or design, N/A (e.g., I am a student), Other industry not listed here]
-

*** End of Survey ***

APPENDIX D

Authorship Statements

Deakin University Authorship Procedure

Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	Losing Confidence in Quality: Unspoken Evolution of Computer Vision Services
Publication details	Presented at the 35th IEEE International Conference on Software Maintenance and Evolution, Cleveland, USA, 2019
Name of executive author	Alex Cummaudo
School/Institute/Division if at Deakin Organisation and address if non-Deakin	Applied Artificial Intelligence Institute
Email or phone	ca@deakin.edu.au

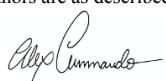
2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?
*If Yes, please complete Section 3
 If No, go straight to Section 4.*

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. <i>(If the same, write "as above")</i>	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Improving Integration and Documentation of Pre-Trained Machine Learning Components
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

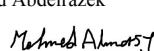
Author 3

John Grundy


Signed: _____
Dated: 22 July 2019

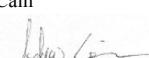
Author 4

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

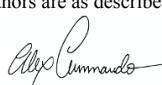
2. Inclusion of publication in a thesis

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: Improving Integration and Documentation of Pre-Trained Machine Learning Components
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: 
 Date:

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).

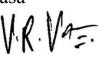
Author 1

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

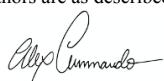
2. Inclusion of publication in a thesis

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: Improving Integration and Documentation of Pre-Trained Machine Learning Components
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	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

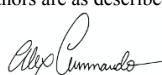
2. Inclusion of publication in a thesis

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: Improving Integration and Documentation of Pre-Trained Machine Learning Components
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: 
 Alex Cummaudo

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

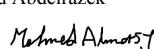
Author 4

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	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

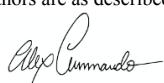
2. Inclusion of publication in a thesis

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: Improving Integration and Documentation of Pre-Trained Machine Learning Components
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

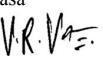
Author 2

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	Requirements of API Documentation: A Case Study into 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

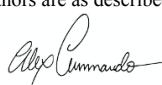
2. Inclusion of publication in a thesis

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: Improving Integration and Documentation of Pre-Trained Machine Learning Components
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: 
 Alex Cummaudo

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

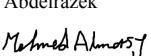
Author 3

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	Manual and Automatic Emotion Analysis of Computer Vision Service Pain-Points
Publication details	Submitted to the 6th International Workshop on Emotion Awareness in Software Engineering
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

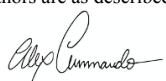
2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis? 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	
Thesis title	Taming the Evolving Black Box: Improving Integration and Documentation of Pre-Trained Machine Learning Components
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: 18 September 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 produced the data set of Stack Overflow posts used for analysis within this paper and contributed to the initial conception of this project. He drafted the methodology section that details how this data set was produced. Additionally, he drafted the threats to validity section, results and discussion sections. He reviewed the entire paper and made contributions to the findings and discussion sections. He assisted in conducting inter-rater reliability with two additional raters (Rajesh and Ulrike Maria). He prepared the graphs and tables, 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 2	Ulrike Maria Graetsch Applied Artificial Intelligence Institute Deakin University
Contribution of author 2	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 3	Maheswaree K Curumsing Applied Artificial Intelligence Institute Deakin University
Contribution of author 3	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 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.

Name and affiliation of author 6

John Grundy
Faculty of Information Technology
Monash University

Contribution of author 6

John Grundy contributed to revisions of the manuscript and guidance for the publication venue. 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: 18 September 2020

Author 2

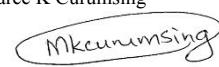
Ulrike Maria Graetsch

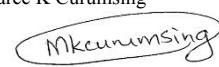


Signed: 
Dated: 18 September 2020

Author 3

Maheswaree K Curumsing



Signed: 
Dated: 18 September 2020

Author 4

Scott Barnett



Signed: 
Dated: 18 September 2020

Author 5

Rajesh Vasa



Signed: 
Dated: 18 September 2020

Author 6

John Grundy



Signed: 
Dated: 18 September 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/semotion21

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

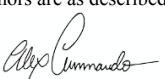
2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis? 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>	Alex Cummaudo
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Improving Integration and Documentation of Pre-Trained Machine Learning Components
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 Ohtake Faculty of Science, Engineering and Built Environment Deakin University
Contribution of author 1	Tomohiro Ohtake 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 Ohtake 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:

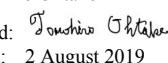
- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these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

Signed: 
Dated: 2 August 2019

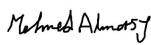
Author 2

Alex Cummaudo


Signed: 
Dated: 2 August 2019

Author 3

Mohamed Abdelrazek


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	Using Pre-Trained Emotion Classification Models on Stack Overflow Questions: Lessons Learned
Publication details	Submitted for the 33rd International Conference on Advanced Information Systems Engineering
Name of executive author	Ulrike Maria Graetsch
School/Institute/Division if at Deakin Organisation and address if non-Deakin	Applied Artificial Intelligence Institute
Email or phone	maria.graetsch@deakin.edu.au

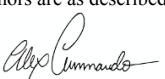
2. Inclusion of publication in a thesis

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>	Alex Cummaudo
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Improving Integration and Documentation of Pre-Trained Machine Learning Components
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 June 2020

4. Description of all author contributions

Name and affiliation of author 1

Ulrike Maria Graestch
Applied Artificial Intelligence Institute
Deakin University

Contribution of author 1

Ulrike Maria's contributed to the initial conception of the project and performed the automatic classifier classifications (EmoTxt) on our Stack Overflow data set, which involved downloading and installing EmoTxt and adapting our data set to be compatible with EmoTxt. Ulrike Maria drafted the initial manuscript, conducted the literature review presented in the work, and performed calculations on the inter-rater agreement statistics. She explored the training dataset of EmoTxt and investigated the data imbalance and emotion labelling bias discussed within the work, and proposal for future tooling to alleviate issues identified.

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 performed a detailed review of the manuscript and made substantial changes to the paper's content, producing figures and tables within the paper. He revised the Fleiss' Kappa statistic and proposed changes to observed percentage agreement. He set up and conducted inter-rater reliability with two additional raters (Maheswaree and Ulrike Maria). He reviewed the entire paper and made contributions to the findings and discussion sections. He validated inter-rater reliability statistics against the three raters and against the automatic classifications made from EmoTxt. He prepared the paper for submission, and ensured the paper was formatted to the guidelines and page limit by reducing whitespace.

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

Maheswaree K Curumsing
Applied Artificial Intelligence Institute
Deakin University

Contribution of author 4

Maheswaree K Curumsing's contribution involved structuring the approach used around the EmoTxt classifier to label emotions within Stack Overflow posts. Further, she contributed to the manual classification for inter-rater reliability. She made further revisions and proofreading to the manuscript and provided modifications

where needed. Maheswaree also contributed in the initial conception of the project by outlining techniques used to run the experiment and her expertise in emotion classification was leveraged in the paper.

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

Ulrike Maria Graetsch



Signed:
Dated: 2 June 2020

Author 2

Alex Cummaudo



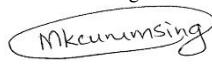
Signed:
Dated: 2 June 2020

Author 3

Rajesh Vasa


Signed:
Dated: 2 June 2020**Author 4**

Maheswaree K Curumsing


Signed:
Dated: 2 June 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/caise21

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