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

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

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

Applied Artificial Intelligence Institute
Deakin University
Melbourne, Australia

January 31, 2020

Abstract

Application developers are eager to integrate machine learning (ML) into their software, with a plethora of vendors providing pre-packaged components—typically under the 'AI' banner—to entice them. Such components are marketed as developer 'friendly' ML and easy for them to integrate (being 'just another' component added to their toolchain). These components are, however, non-trivial: in particular, developers unknowingly add the risk of mixing nondeterministic ML behaviour into their applications that, in turn, impact the quality of their software. Prior research advocates that a developer's conceptual understanding is critical to effective interpretation of reusable components. However, these ready-made AI components do not present sufficient detail to allow developers to acquire this conceptual understanding. In this study, by use of a mixed-methods approach of survey and action research, we investigate if the application developers' deterministic approach to software development clashes with the mindset needed to incorporate probabilistic components. Our goal is to develop a framework to better document such AI components that improves both the quality of the software produced and the developer productivity behind it.

Declarations

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

(i) I am the creator of all or part of the whole work(s) (including content and layout) and that where reference is made to the work of others, due acknowledgment 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.

Alex Cummaudo
BSc *Swinburne*, BIT(Hons)
January 31, 2020

To my family, friends, and teachers — all this worthless and impossible without you.

Acknowledgements

A long journey of 20 years education has led me to this thesis and upon reflection, there are so many people that have helped get me to this point. To start, I must thank my family for your support. Each of you have always been there for me in times good and bad. I'm especially grateful to my mother, Rosa, and father, Paul, for your love and support, and to my nieces Nina and Lucy—though too young to read this now—for bringing us all so much joy in the 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 journey. 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 helped rewire my thinking and worldview to far wider perspectives and 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, your interest in my work, and the joint collaborations we achieved in the last two years; Associate Professor Mohamed Abdelrazek for your help in shaping many of the works within this thesis; and, lastly, Associate Professor Andrew Cain, who not only taught me the realm of programming back in undergrad days, but who gave me the support that a PhD was within my reach, of which I had never fathomed. I must thank everyone at the Applied Artificial Intelligence Institute for creating such an enjoyable environment to work in, especially Jake Renzella, Rodney Pilgrim, Mahdi Babaei, and Reuben Wilson for their friendship over these years and for all the coffee runs, conversations, and ideas shared. And, lastly, to Tom Fellowes: thank you for being by my side throughout this journey.

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

— Alex Cummaudo
July 2020

Contents

List of Publications

Below lists publications arising from work completed in this PhD.

1.
2.
3.
4.
5.
6.
7.
8.

List of Figures

List of Tables

List of Listings

Part I

Preface

CHAPTER 1

Introduction

⟨ TODO: *Introduce a broad statement of the problem. AI-first software is data-driven, but documentation and nature of services that provide these services make it look like they are deterministic* ⟩

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

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

1. The application developer writes an ML classifier from scratch and trains it from a handcrafted and curated dataset. This approach is laborious in time and demands formal training in ML and mathematical knowledge, but the tradeoff is that they have full autonomy in the models they creates.
2. The application developer downloads a pre-trained model and 'plugs' it into an existing ML framework, such as Tensorflow [**?**]. While this approach is less demanding in time, it requires them to revise and understand how to 'glue'

[1] A **?** report by market research company Forrester captured such growth into four key areas [**?**], as reproduced in **??**.

components of the ML framework together[2] into their application's code.

3. The application developer uses a data-driven and cloud-based service. They don't need to know anything behind the underlying 'intelligence' and how it functions. It is fast to integrate into their applications, and the application programming interfaces (APIs) offered abstracts the technical know-how behind a web call.

The documentation of the service alludes that the data-driven service is as similar to other cloud services offered by the provider. Because this is 'another' cloud service, the application developer *assumes* it would act and behave as any other typical service would. But does this assumption—and a lack of appreciation of ML—lead to developer pain-points and miscomprehension? If so, how can the service providers improve their documentation to alleviate this? Do these data-driven services share similarities to the runtime behaviour of traditional cloud services? And if not, how best can the application developer integrate the data-driven service into their a rule-driven application to produce AI-first software?

?? provides an illustrative overview between the context clashing of rule-driven applications and data-driven cloud services, and we contrast characteristics of typical cloud systems and data-driven ones in **??**.

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

1.1 Research Context

⟨ TODO: *Unpack the 'spectrum' a little, and make it clear that this thesis focuses on computer vision services.* ⟩

As described, the application developer has three key approaches in producing AI-first software. This 'range' of AI-first integration techniques partially reflects Google AI's[3] *machine learning Spectrum* [**? ? ?**], which encompasses the variety of skill, effort, users and types of outputs of integration techniques. One extreme involves the academic research of developing algorithms and self-sourcing data to

[2]Thus 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.

[3]Google 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* [**?**] is evident. A further example includes how Facebook leverage AI *at scale* within their infrastructure and platforms [**?**].

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

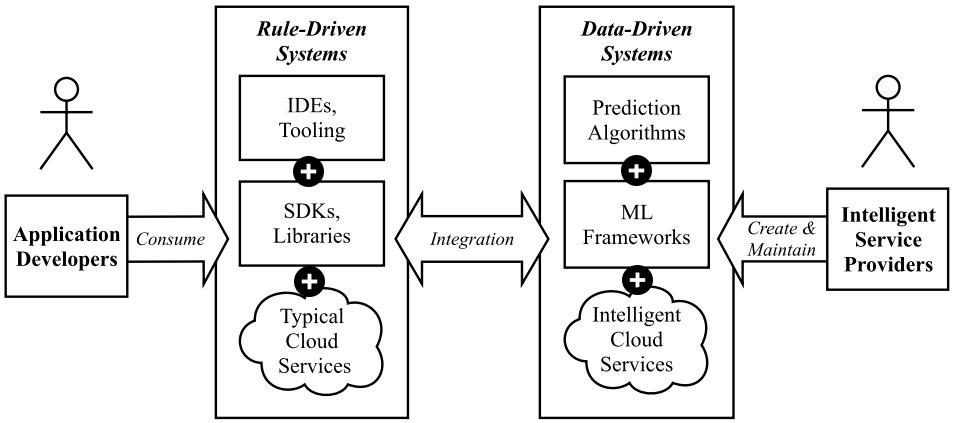
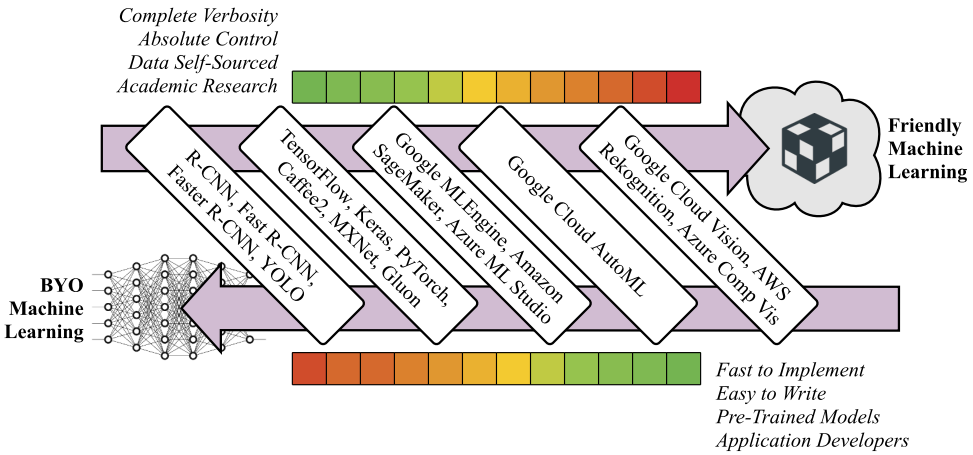


Figure 1.2: Examples within the machine learning spectrum of computer vision. Colour scales indicates the benefits (green) and drawbacks (red) of each end of the spectrum.



achieve intelligence—coined as Build Your Own Machine Learning (BYOML) [**? ? ?**]. 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 predictive intelligence. We illustrate the slightly varied characteristics within this spectrum in **??** and **??**.

These data-driven 'friendly' services are gaining traction within developer circles: we show an increasing trend of Stack Overflow posts mentioning a mix of intelligent computer vision (CV) services in **??**.[4] Academia provides varied nomenclature for these services, such as *Cognitive Applications* and *Machine Learning Services* [**?**] or *Machine Learning as a Service* [**?**]. For the context of this

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

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

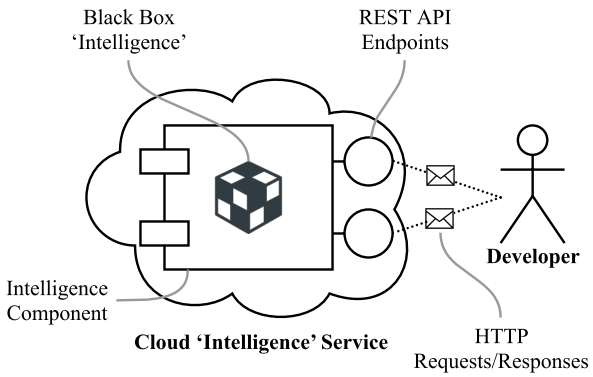
intelligent web service	**Typical web service**
Probabilistic	Deterministic
Machine Learnt	Human Engineered
Data-Driven	Rule-Driven
Black-Box	Mostly Transparent

Table 1.2: Comparison of the machine learning spectrum.

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

thesis, we will refer to such services under broader term of **intelligent web services (IWSs)**, and diagrammatically express their usage within **??**.

Figure 1.3: Overview of IWSs.



While there are many types of IWSs available to software developers,[5], 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 (typically) available as a cloud-based Platform as a Service (PaaS).[6,7] For a given input, developers receive an 'intelligent' response and an associated confidence value that represents the likelihood of that result. This is typically serialised as a JSON/XML response object.

> 👉 *Within this thesis, we scope our investigation to a mature* subset *of IWSs that provide computer vision intelligence [**? ? ? ? ? ? ? ? ? ? ? ? ? ?**]. For the context of this thesis, we will refer to such services as **computer vision services (CVSs)**.*

1.2 Research Motivation

⟨ TODO: *This section describes deterministic mindset clashing against probabilistic behaviour* ⟩

⟨ TODO: *MA: It is useful to highlight the takeaways of each paragraph - e.g. probabilistic nature of ML, Choosing a threshold problem, lack of understanding*

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

[6]We 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.

[7]A 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.

of how ml works, documentation style of ml models. then at the end of the section, clearly state the problem/gap considered ⟩

IWSs are accessible through APIs consisting of 'black box' intelligence (**??**).[8] Data scientists produce ML algorithms to make predictions in our datasets and discover patterns within them. These black boxes are inherently probabilistic and stochastic; there is little room for certainty in these results, as the insight is purely statistical and associational [**?**] against its training dataset.

Evermore applications are using CVSs as demonstrated by ubiquitous examples: aiding the vision-impaired [**?** **?**], accounting [**?**], data analytics [**?**], and student education [**?**]. However, developers who build these applications do not need to treat their programs in a stochastic or probabilistic mindset manner, given a rule-driven mindset that computers make certain outcomes. A CVS, for example, may return the *probability* that a particular object exists in an input images' pixels, and thus for a more certain (though not fully certain) distribution of overall confidence returned from the service, a developer must treat the problem stochastically by testing this case hundreds if not thousands of times to find a richer interpretation of the inference made.

Unless a change in API version occurs, educators teach application developers a deterministic mindset that an API call returns the same outputs for the same inputs. As an example, consider simple arithmetic representations (e.g., $2 + 2 = 4$). The deterministic (rule-driven) mindset suggests that the result will *always* be 4. However, the non-deterministic (data-driven) mindset suggests that results are probable: target output (*exactly* 4) and the output inferred (*a likelihood of* 4) matches as a probable percentage (or as an error where it does not match).[9] Instead of an exact output, there is a *probabilistic* result: $2 + 2$ *may* equal 4 to a confidence of n.

Developers must appreciate such a critical principle of ML software when building quality AI-first applications, particularly reliability. The external quality of 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." [**?**]. 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. 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 [**?** **?**] and providers should consider mismatches between a developer's conceptual knowledge of the API its implementation [**?**]. Unreliable APIs ultimately hinder developer performance and thus reduces productivity. **??**

[8] The '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 [**?** **?**] to describe "systems whose internal mechanisms are not fully open to inspection" [**?**].

[9] **?**] produces a multi-layer perceptron neural network performing arithmetic representation.

provides two motivating scenarios describing these concerns in greater detail.

1.2.1 Motivating Scenarios

⟨ TODO: *Could move this into the appendix to keep introduction chapter tighter* ⟩
⟨ TODO: *JG: or could use one of these at start of section* 1.2 *to help clarify/motivate...* ⟩
⟨ TODO: *yes too long in this chapter - use one, simplify a bit. Could put other one in appendix...* ⟩
⟨ TODO: *MA: i guess the discussion related to Table*1.3 *and Table*1.4 *could be moved to* 1.2 *as one of the challenges* ⟩ ⟨ TODO: *MA: after reading the section, it looks like you could move it before* 1.2 ⟩

The market for IWSs is increasing (**??**) and as is developer uptake and enthusiasm in the software engineering community (**??**). However, the impact to software quality (internal and external) due to a mismatch of the application developer's deterministic mindset and the service provider's nondeterministic mindset is of concern.

To illustrate the context of use, we present the two scenarios of varying risk: (i) a fictional software developer, named Tom, who wishes to develop an inherently low-risk photo detection application for his friends and family; and (ii) a high-risk cancer clinical decision support system (CDSS) that uses patient scans to recommend if surgeons should send their patients to surgery. Both describe scenarios where AI-first components has substantiative impact to end-users when the software engineers developing with them misunderstand the nuances of ML, ultimately adversely affecting external quality. Moreover, due to lack of comprehension, this hinders developer experience, productivity, and understanding/appreciation of AI-based components.

Motivating Scenario I: Tom's *PhotoSharer* App

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

Instead of building a computer vision engine from scratch, which takes too much time and effort, Tom thinks he can achieve this using one of the common CVSs. Tom comes from a typical software engineering background and has insufficient knowledge of key computer vision terminology and no understanding of its underlying techniques. However, inspired by easily accessible cloud APIs that offer computer vision analysis, he chooses to use these. Built upon his experience of using other similar cloud services, he decides on one of the CVS APIs, and expects a static result always and consistency between similar APIs. Analogously, when Tom invokes the iOS Swift substring method `"doggy".prefix(3)`, he expects it to be consistent with the Android Java equivalent `"doggy".substring(0, 2)`. Consistent, here, means two things: (i) that calling `substring` or `prefix` on `'dog'` will *always*

return in the same way every time he invokes the method; and (ii) that the result is *always* `'dog'` regardless of the programming language or string library used, given the deterministic nature of the 'substring' construct (i.e., results for substring are API-agnostic).

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

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

More concretely, in **??**, we illustrate how three (anonymised) CVS providers fail to provide similar consistency to that of the substring example above. If Tom uploads a photo of a border collie[10] to three different providers in August 2018 and January 2019, he would find that each provider is different in both the vocabulary used between. The confidence values and labels within the *same* provider varies within a matter of five months. The evolution of the confidence changes is not explicitly documented by the providers (i.e., when the models change) nor do they document what confidence means. Service providers use a tautological nature when defining what the confidence confidence values are (as presented in the API documentation) provides no insight for Tom to understand why there was a change in confidence, which we show in **??**, unless he *knows* that the underlying models change with them. Furthemore, they do not provide detailed understanding on how to select a threshold cut-off for a confidence value. Therefore, he's left with no understanding on how best to tune for image classification in this instance. The deterministic problem of a substring compared to the nondeterministic nature of the IWS is, therefore, non-trivial.

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

- What does 'confidence' mean?

[10]The image used for these results is `https://www.akc.org/dog-breeds/border-collie/`.

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

API Provider	Definition(s) of Confidence
Provider A	*"Score is the confidence score, which ranges from 0 (no confidence) to 1 (very high confidence)."* [**?**]
	"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]." [**?**]
	"The overall score of the result. Range [0, 1]" [**?**]
Provider B	*"Confidence score, between 0 and 1... if there insufficient confidence in the ability to produce a caption, the tags maybe [sic] the only information available to the caller."* [**?**]
	"The level of confidence the service has in the caption." [**?**]
Provider C	*"The response shows that the operation detected five labels (that is, beacon, building, lighthouse, rock, and sea). Each label has an associated level of confidence. For example, the detection algorithm is 98.4629% confident that the image contains a building."* [**?**]
	"[Provider C] also provide[s] a percentage score for how much confidence [Provider C] has in the accuracy of each detected label." [**?**]

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

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

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

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

Motivating Scenario II: Cancer Detection CDSS

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

In **??** 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.

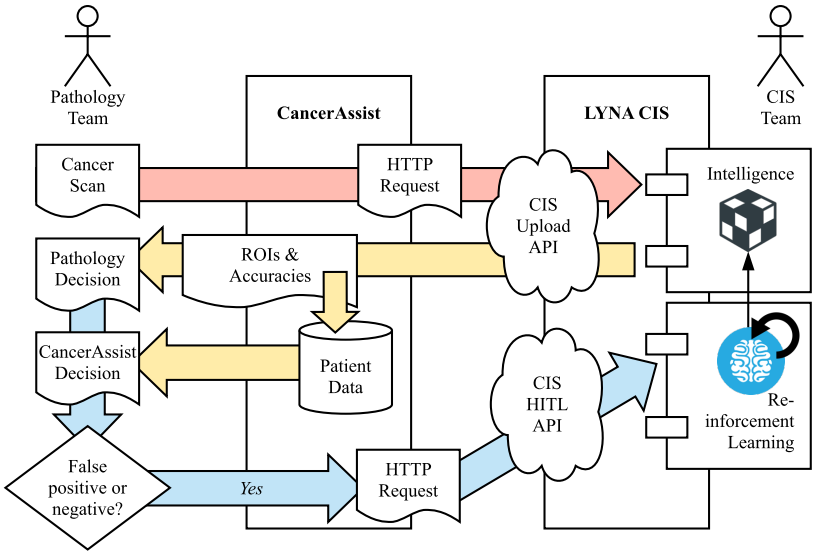


Figure 1.4: CancerAssist Context Diagram. ***Key:*** *Red Arrows = Scan Input; Yellow Arrows = Decision Output; Blue Arrows = HITL Feedback Input.*

Powering CancerAssist is Google AI's Lymph Node Assistant (LYNA) [**?**],

a CNN based on the Inception-v3 model [**?** **?**]. 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 are 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 as CancerAssist and its underlying IWS. The ramifications of not being able to test such a system and putting it out into production may prove fatal to patients.

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

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

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

1.3 Research Goals

1.3.1 Research Overview

⟨ TODO: *is it JUST CVS, or they one exemplar of intelligent APIs/services??* ⟩
This thesis aims to investigate and better understand the nature of cloud-based computer vision services (CVSs).[11] 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 [**?**] and Boehm [**?**] 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 **??**.) 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)

[11] As 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.

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

with six sub-RQs, based on both empirical and non-empirical software engineering methodology [**? ?**], further discussed in **??**.

1.3.2 Research Questions

Firstly, we investigate adverse implications that arise when using CVSs that affects consistency and robustness (**??**). 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. Thus, we conclude how the nature of these services (at present) are not fully robust, consistent, and thus not reliable. Formally, we structure the following RQs:

> **RQ1. What is the nature of cloud-based CVS?**
>
> *RQ1.1.* What is their runtime behaviour?
> *RQ1.2.* What is their evolution profile?

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

> **RQ2. How complete is current CVS APIs documentation?**
>
> *RQ2.1.* What are the dimensions of '*complete*' API documentation, according to both literature and practitioners?
> *RQ2.2.* What additional information or attributes do developers need in CVS API documentation to make it more complete?

Thirdly, we investigate how software developers approach using these services and directly assess developer pain-points resulting from the nature of CVSs and their documentation (**??**). We show that there is a statistically significant difference in these complaints when contrasted against more established software engineering domains (such as web or mobile development) as expressed as posts on Stack Overflow. We provide a number of exploratory avenues for researchers, educators, software engineers and IWS providers to alleviate these complaints based on this analysis. We formulate the following RQs:

> **RQ3. How does the developer's comprehension of CVSs differ to conventional software engineering domains?**
>
> *RQ3.1.* What aspects of CVSs and its documentation do developers struggle with, as expressed as pain-points within Stack Overflow posts?

RQ3.2. Is the distribution of the struggles within these posts different to established computer vision service fields?

RQ3.3. Are developers frustrated with CVSs? What emotions do they express?

Lastly, based on our analysis of the documentation, runtime and evolution profiles of CVSs, in conjunction with our assessment of developer complaints, we developed two specific solutions to address documentation weaknesses and another to ensure greater reliability when developing using CVS (**????**). Our first solution, via synthesis of a documentation template, proposes recommendations on where current CVS documentation needs improvement to make the documentation more reliable. Our second solution consists of an integration architecture style (or facade) that guards against the runtime and evolution issues identified of CVSs to make integrating with a CVS more reliable. Our final RQ is:

RQ4. What strategies can developers use to integrate with and use CVSs, while preserving robustness and reliability?

⟨ TODO: *you might want to include a section on your methodology here - so the flow is like questions and gaps, methodology, and contributions* ⟩

1.4 Research Methodology

⟨ TODO: *Short section on research methodology... greater described in chapter* ⟩

1.5 Thesis Organisation

We organise the thesis into four parts. **??, (*The Preface*)** includes introductory, background and methodology chapters. This is a *PhD by Publication*, and **?? (*Publications*)** comprises of seven publications resulting from this work over **??????????????**; publications are included verbatim except for terminology and formatting changes to better fit the suitability of a coherent thesis. **??, (*The Postface*)** includes the conclusion and future works chapter, as well as a list of academic studies and online artefacts referenced within the thesis. **??, (*Appendices*)** includes all supplementary material, including mandatory authorship statements and ethics approval. Details of each chapter following this introductory chapter are provided in the following section.

1.5.1 Overview of Chapters

??: Preface

⟨ TODO: *CHECK THE RULES for thesis by publication - are you allowed these exta chapters?? Else intgrate into this one...* ⟩

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

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

??: Publications

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

??: Investigating improvements to CVS API documentation This chapter was originally a short paper presented at the 2019 International Symposium on Empirical Software Engineering and Measurement (ESEM) [**?**]. To understand where to improve CVS documentation, we first need to investigate *what* makes a good API document. This short paper initially answered one aspect of **??**: what *academic literature* suggests a good (complete) API document should comprise of. By conducting an systematic mapping study resulting in 21 primary studies, we systematically develop a taxonomy that combines the recommendations of scattered work into a structured framework of 5 dimensions and 34 weighted categorisations. We then extend this work[13] by triangulating the taxonomy with opinions from developers using the System Usability Scale to assess the efficacy of these recommendations (thereby answering the second aspect of **??**). From this, we assess the how well CVS providers document their APIs via a heuristic validation of the taxonomy, using the three services from the ICSME publication to make recommendations where documentation should be more complete, thereby answering **??** (and thus **??**).

[13]The extended version of this chapter has been submitted to ⟨ TODO: *Revise the Journal of Systems and Software*⟩ in [**?**] and is currently within review.

??: Understanding developer struggles when using CVSs This chapter has been accepted for presentation at the 2020 International Conference on Software Engineering (ICSE) [**?**]. We conduct a mining study of 1,425 Stack Overflow posts that provide indications of the types frustrations that developers face when integrating CVSs into their applications. To gather what their pain-points are, we use two classification taxonomies that also use Stack Overflow to understand generalised and documentation-specific pain-points in mature software engineering (SE) domains. This study answers **??** in detail and provides a validation to our motivation of **??**: we validate that the *completeness* of current CVS API documentation is a main concern for developers and there is insufficient explanation into the errors and limitations of the service. We find that the documentation does not adequately cover all aspects of the technical domain. In terms of integrating with the service, developers struggle most with simple errors and ways in which to use the APIs; this is in stark contrast to mature software domains. Our interpretation is that developers fail to understand the IWS lifecycle and the 'whole' system that wraps such services. We also interpret that developers have a shallower understanding of the core issues within CVSs (likely due to the nuances of ML as suggested in a discussion in the paper), which warrants an avenue for future work in software engineering education.

??: Interpreting developer emotions per question category This chapter has been submitted to the the 2020 International Workshop on Emotion Awareness in Software Engineering (SEmotion) [**?**].

??: Interpreting developer emotions per question category This chapter was presented at the 2019 International Conference on Web Engineering (ICWE) [**?**].

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

??: Developing a CVS integration architecture This chapter has been submitted to the 2020 Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering [**?**].

??: Postface

?? - Conclusions In this chapter, we review the contributions made in this thesis and the relevance and significance to identifying and resolving key issues when application developers integrate with CVS. We evaluate these outcomes with reference to the research goals, and discuss threats to validity of the work. Lastly, we discuss the various avenues of research arising from this work.

??: Appendices

?? provides additional material referenced within this thesis but not provided in the body. We provide mandatory coauthor declaration forms describing the contribution breakdown for each publication within **??**. **??** contains copies of the ethics clearance for various experiments within this thesis. We describe the list of primary sources arising in the systematic mapping study we conduct in **??** within **??**.

1.5.2 Coherency of Publications

In this section, we detail how each publication forms a coherent body of work. After our exploratory analysis on the nature of CVSs (**??**), we proposed two sets of recommendations targeted towards two CVS stakeholders: (i) CVS consumers (i.e., application developers) and (ii) CVS providers. Our subsequent publications arose as a two-fold investigation to develop two strategies in which developers and providers can, respectively, (i) better integrate CVSs into their applications, and (ii) how CVSs can be better documented. **??** provides a tabulated form of the publications and research questions addressed within this thesis. We also provide abbreviations for easier reference in this section. A high-level overview of the cohesiveness of our publications is provided in **??**.

Landscape Analysis and Preliminary Solution Design

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

The ICSME paper confirmed our hypotheses that the services have a non-deterministic behavioural profile, and that the evolution occurring within the ML

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

[16]Date of publication, if applicable.

[17]The extended version of this conference proceeding is provided in **??**.

[18]We abbreviate this with an added 'd' (for the demonstrations track) to distinguish this paper from our full ICSE 2020 paper.

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

Ref.	Venue	Acronym	Rank[14]	Published[15]	Chapter	RQs
[?]	35th International Conference on Software Maintenance and Evolution	ICSME	A	05 Dec 2019	??	??
[?]	13th International Symposium on Empirical Software Engineering and Measurement	ESEM	A	17 Oct 2019	Excluded[16]	??
[?]	Journal of Systems and Software	JSS	Q1	*In Press*	??	??
[?]	42nd International Conference on Software Engineering	ICSE	A*	*In Press*	??	??
[?]	5th International Workshop on Emotion Awareness in Software Engineering[17]	SEmotion	A*	*In Press*	??	??
[?]	13th International Conference on Web Engineering	ICWE	B	26 Apr 2019	??	??
[?]	42nd International Conference on Software Engineering	ICSE(d)[18]	A*	*In Press*	??	??
[?]	28th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	ESEC/FSE	A*	*In Press*	??	??

models powering these services are not sufficiently communicated to software engineers. This therefore led to follow up investigation into how developers perceive these services, and thereby determine if they are frustrated due to this lack of communication.

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

Improving Documentation Attributes

As mentioned, our ICSME paper found that evolutionary and non-deterministic behavioural profile of are not adequately documented in the service's APIs documentation. A recommendation concluding from this work was that service providers should improve their documentation, however there lacked a strategy by which they could do this, and our hypotheses that developers were actually frustrated by this

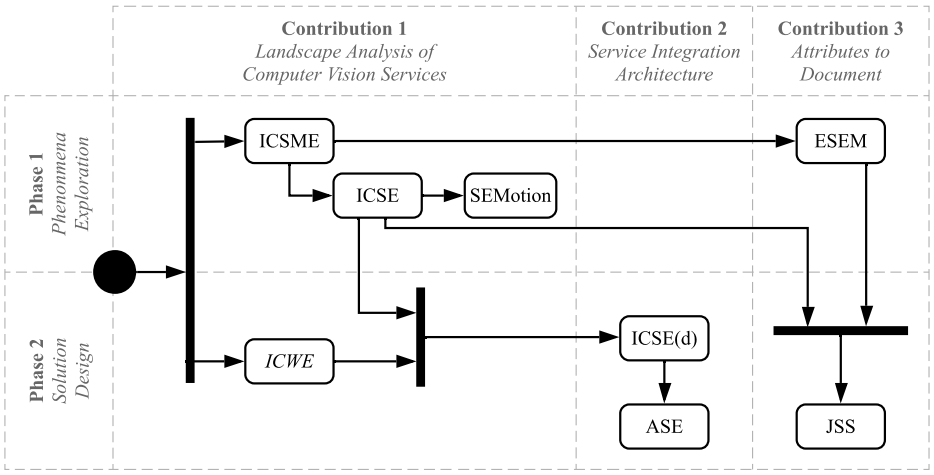


Figure 1.5: Activity diagram of the coherency of our publications, how our research was conducted, and relevant connections between publications. Our two-phase structure initial phenomena exploration and a proposed solutions to issues identified from the exploration. We map the contributions within each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

lack of communication was yet to be tested. This led to two follow-up further investigations as presented in our ICSE and ESEM papers.

One aspect of our ICSE paper was to confirm whether developers are actually frustrated with the service's limited API documentation. By mining Stack Overflow posts with reference to documentation issues, we adopted a **?** documentation-related taxonomy by **?**] to classify posts, and found that 47.87% of posts classified fell under the 'completeness' dimension of **?** 's taxonomy. This interpretation, therefore, warranted the recommendation proposed in the ICSME paper to improve service documentation.

However, though improvements to more complete documentation was justified from the ICSE paper, we needed to explore exactly *what* makes a 'complete' API document. By conducting a systematic mapping study resulting in 4,501 results, we curated 21 primary studies that outline the facets of API documentation knowledge. From these studies, we distilled a documentation framework describing a prioritised order of the documentation assets API's should document that is described in our ESEM short paper. After receiving community feedback, we extended this short paper with a follow-up experiment submitted to JSS. By conducting a survey with developers, we assessed our API documentation taxonomy's efficacy with practitioner opinions, thereby producing a weighted taxonomy against *both* literature and developer sources. Lastly, we triangulated both weightings against a heuristic evaluation against common CVS providers' documentation. This allowed us to deduce which specific areas in existing CVS providers' API documentation needed improvement, which was a primary contribution from our JSS article.

Service Integration Architecture

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

We proposed two technical solutions in ICSE(d) and ESEC/FSE, respectively, to help alleviate this issue. ⟨ TODO: *Revise this... needs to be fleshed out* ⟩ Firstly, our ICSE(d) paper provides a workflow for developers to better select an appropriate confidence threshold, and thus decision boundary, calibrated for their particular use case. In our ESEC/ASE paper, we provide a reference architecture for developers to guard against the non-deterministic issues that may 'leak' into their applications. This architecture is a facade style, similar to the style proposed in our ICWE paper, however, unlike the ICWE paper that uses proportional representation approach to modify multiple sources, our ESEC/FSE paper proposes a guarded relay, whereby a single service is used, and the facade should maintain a lifecycle to monitor evolution issues identified in ICSME and should be benchmarked against the developer's dataset (i.e., against the particular application domain) as suggested in ICSE(d). These two primary contributions further serve as an answer to **??**.

1.6 Research Contributions

⟨ TODO: *would be good to have more detail HERE on the key contributions made in this work* ⟩

⟨ TODO: *Please include more details of the contributions - e.g. which research question is addressed in this contribution, key findings, you could also list the publications associated with each one. Also in this case you do not need section* 1.4.2 ⟩

The outcomes of answering these four primary research questions helps shape the three primary contributions this thesis offers to software engineering knowledge:

- An improved understanding in the landscape of CVSs, with respect to their runtime behaviour and evolutionary profiles.
- A service integration architecture that helps developers with integrating their applications with CVSs.

- A list of attributes that should be documented to assist CVS providers to better document their services.

These contributions are detailed further within **??**. ⟨ *TODO: I wonder if putting summary of key thesis contributions HERE to end the chapter might be best*??? ⟩

CHAPTER 2

Background

In **??**, we defined a common set of (artificial) intelligence-based cloud services that we label intelligent web services (IWSs). Specifically, we scope the primary body of this study's work on computer vision services (CVSs) (e.g., Google Cloud Vision [**?**], AWS Rekognition [**?**], Azure Computer Vision [**?**], Watson Visual Recognition [**?**] etc.). We claim developers have a distinctly deterministic mindset (2 + 2 *always* equals 4) whereas an IWS's 'intelligence' component (a black box) may return probabilistic results (2 + 2 *might* equal 4 *with a confidence of* 95%). Thus, there is a mindset mismatch between probabilistic results (from the API provider) and results interpreted with certainty (from the API consumer).

What affect does this mindset mismatch have on the developer's approach towards building probabilistic software? What can we learn from common software engineering practices (e.g., [**? ?**]) that apply to resolve this mismatch and thereby improve quality, such as verification & validation (V&V)? Chiefly, we anchor this question around three lenses of software engineering: creating an IWS, using an IWS, and the nature of IWSs themselves.

Our chief concern lies with interaction and integration between IWS providers and consumers, the nature of applications built using an IWS, and the impact this has on software quality. We triangulate this around three pillars, which we diagrammatically represent in **??**.

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

(2) The usage of the IWS. We investigate the external quality attributes of using an IWS from the IWS *consumer's* perspective. That is, we ask if existing validation techniques are sufficient enough to ensure that the end-users can

Pillar/Perspective	*Development of an IWS*	*Usage of an IWS*	*Nature of an IWS*
Software Quality	X	X	X
Probabilistic & Non-Deterministic Systems	X	X	X
APIs & their usability	X	X	X

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

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

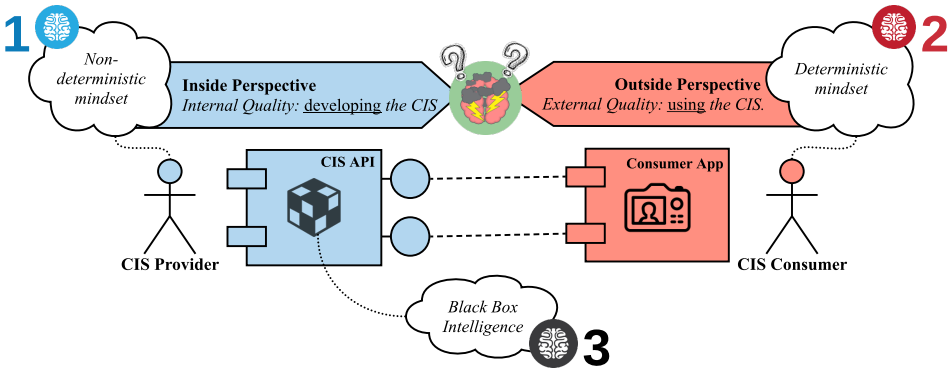


Figure 2.1: The three pillars by which we anchor the background: (1) developing an IWS with a non-deterministic mindset by the IWS provider; (2) the use of a IWS with a deterministic mindset by the IWS consumer; (3) the nature of a IWS itself.

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

2.1 Software Quality

> *Quality... you know what it is, yet you don't know what it is.*
>
> Robert Pirsig, 1974 [**?**]

The philosophical viewpoint of 'quality' remains highly debated and there are multiple facets to perceive this complex concept [**?**]. Transcendentally, a viewpoint like that of **?** 's above shows that quality is not tangible but still recognisable; it's hard to explicitly define but you know when it's missing. The **?** provides a breakdown of

seven universally-applicable principles that defines quality for organisations, developers, customers and training providers [**?**]. More pertinently, the **?** ISO standard for quality was simply "the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs" [**?**].

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

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

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

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

2.1.1 Validation and Verification

To explain V&V, we analogously recount a tale given by **?**] on his works on reliability. A high-school student sat a standardised test that was sent to 350,0000 students [**?**]. A multiple-choice algebraic equation problem used a variable, a, and intended that students *assume* that the variable was non-negative. Without making this assumption explicit, there were two correct answers to the multiple choice answer. Up to 45,000 students had their scores retrospectively boosted by up to 30 points for those who 'incorrectly' answered, however, outcomes of a student's

higher education were, thereby, affected by this one oversight in quality assessment. The examiners wrote a poor question due to poor process standards to check if their 'correct' answers were actually correct. The examiners "didn't build the right product" nor did they "build the product right" by writing an poor question and failing to ensure quality standards, in the phrases **?**] coined.

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

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

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

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

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

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

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

assured that what they enter in a testing phase outcome the same result when in production?

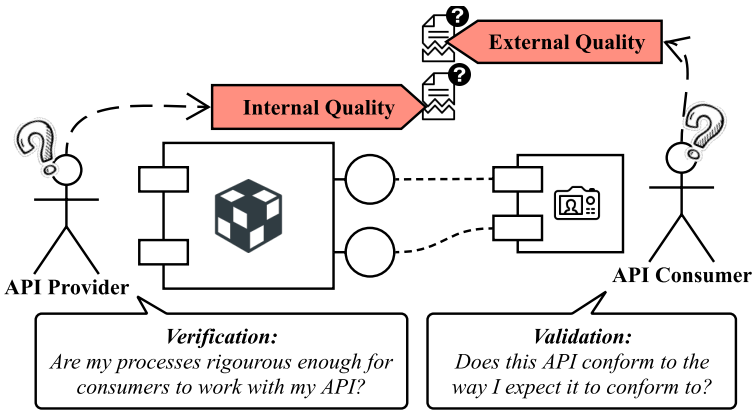


Figure 2.2: The 'leakage' of internal quality into the API consumer's product and external quality imposing on the API provider.

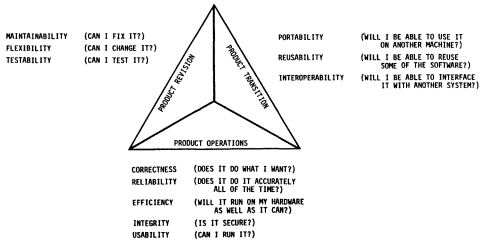
Therefore, just as the test answers with were both correct and incorrect at the same time, so is the same with IWSs returning a probabilistic result: no result is certain. While V&V has been investigated in the area of mathematical and earth sciences for numerical probabilistic models and natural systems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

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

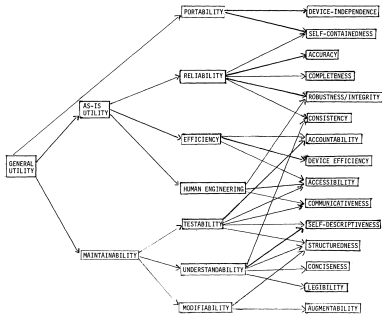
2.1.2 Quality Attributes and Models

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

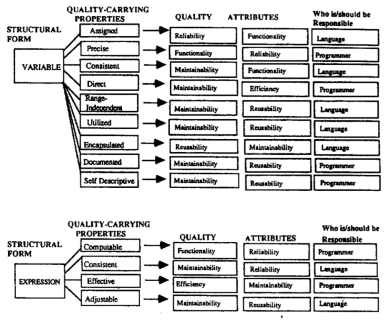
Quality attribute assessment models (like those shown in **??**) are an early concept in software engineering, and systematically evaluating software quality appears as early as **?** [**?**]. **?** 's **?** study introduced the phrase 'attributes' as a "prose



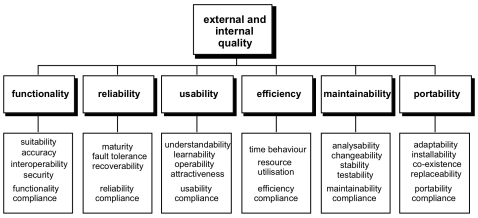
(a) McCall's quality software factors (**?**) [**?**].



(b) Bohem's software quality characteristics tree (**?**) [**?**].



(c) Dromey's quality-carrying properties and programming languages (**?**) [**?**].



(d) ISO/IEC software product evaluation characteristics (**?**) [**?**].

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

expression of the particular quality of desired software" (as worded by **?**]) and 'metrics' as mathematical parameters on a scale of 0 to 100. Early attempts to categorise wider factors under a framework was proposed by **?** in the late 1970s [**?**]. 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?*) (**??**). The model also introduced 11 attributes alongside numerous direct and indirect measures to help quantify quality. This model was further developed by **?**] 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 **??**. In the mid-1990s, Dromey's interpretation [**?**] defined a set of quality-carrying properties with structural forms associated to specific programming languages and conventions (**??**). The model also supported quality defect identification and proposed an improved auditing method to automate defect detection for code editors in IDEs. As the need for quality models became prevalent, the **?** standardised software quality under ISO/IEC-9126 [**?**] (the Software Product Evaluation Characteristics, **??**), which has since recently been revised to ISO/IEC-25010 with the introduction of the Systems and software Quality Requirements and Evaluation (SQuaRE) model [**?**], separating quality into *Product Quality* (consisting of eight quality characteristics and 31 sub-

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

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

?	Extent to which a program can be expected to perform its intended function with required precision [**?**].
?	Code possesses the characteristic *reliability* to the extent that it can be expected to perform its intended functions satisfactorily [**?**].
?	Functionality implies reliability. The reliability of software is therefore dependent on the same properties as functionality, that is, the correctness properties of a program [**?**].
ISO/IEC-9126	The capability of the software product to maintain a specified level of performance when used under specified conditions [**?**].

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

Moreover, defining these systems in themselves is challenging when requirements specifications and solution descriptions are dependent on nondeterministic and probabilistic algorithms. We discuss this further in **??**.

2.1.3 Reliability in Computer Vision

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

[1] In 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.

images are slightly perturbed to maximise prediction error but are still interpretable to humans. Refer to **??**.

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

Thus, the state-of-the-art computer vision techniques may not be reliable enough for safety critical applications (such as self-driving cars) as they do not handle intentional or unintentional adversarial attacks. Moreover, as such adversarial examples exist in the physical world [**?** **?**], "the real world may be adversarial enough" [**?**] to fool such software.

2.2 Probabilistic and Nondeterministic Systems

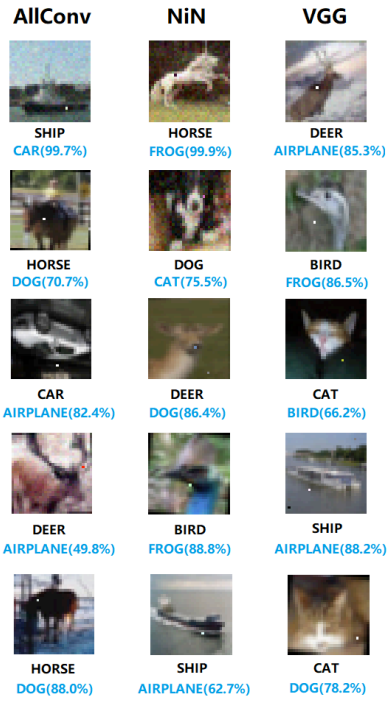
Probabilistic and nondeterministic systems are those by which, for the same given input, different outcomes may result. The underlying models that power an IWS are treated as though they are nondeterministic; **??** introduces IWSs as essentially black-box behaviour that can change over time. As such,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

2.2.1 Interpreting the Uninterpretable

As the rise of applied AI increases, the need for engineering interpretability around models becomes paramount, chiefly from an external quality perspective that the *reliability* of the system can be inspected by end-users. Model interpretability has been stressed since early machine learning research in the late 1980s and 1990s (such as **?**] and **?**]), and although there has since been a significant body of work in the area [**?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?**], it is evident that 'accuracy' or model 'confidence' is still used as a primary criterion for AI evaluation [**?** **?** **?**]. Much research into neural network (NN) or support vector machine (SVM) development stresses that 'good' models are those with high accuracy. However, is accuracy enough to justify a model's quality?

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

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



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

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

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

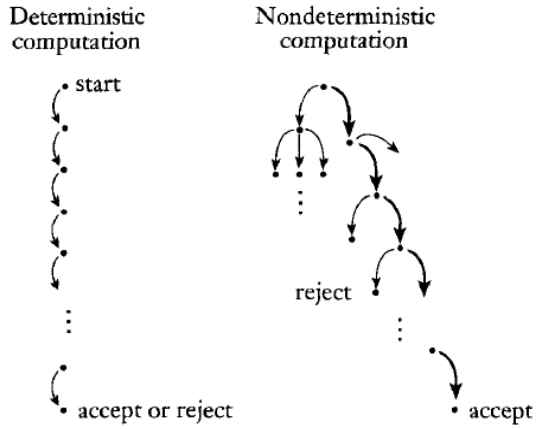


Figure 2.5: A deterministic system (left) always returns the same result in the same amount of steps. A nondeterministic system does not guarantee the same outcome, even with the same input data. Source: [**?**].

from past data" [**?**], a problem commonly referred to as the *datashift* problem [**?**]. Analogously, **?**] 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.*** [**?**]

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 [**?**]), what is known is (i) there exists a critical trade-off between accuracy and interpretability [**? ? ? ? ? ?**], and (ii) a single quantifiable value cannot satisfy the subjective needs of end-users [**?**]. As ever-growing domains ML become widespread[2], 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 [**? ? ?**]), 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 behind explanations[3]) are not realised: model optimisation is only worthwhile if

[2]In areas such as medicine [**? ? ? ? ? ? ? ? ? ? ? ?**], bioinformatics [**? ? ? ? ?**], finance [**? ? ?**] and customer analytics [**? ?**].

[3]*Interpretations* and *explanations* are often used interchangeably.

they "actually solve the original [human-centred] task of providing explanation" [**?**] to end-users. **Therefore, when human-decision makers must be interpretable themselves [?], any AI they depend on must also be interpretable.**

Recently, discussion behind such a notion to provide legal implications of interpretability is topical. **?**] discuss when explanations are not provided from a legal stance—for instance, those affected by algorithmic-based decisions have a 'right to explanation' [**? ?**] under the European Union's GDPR[4]. But, explanations are not the only way to ensure AI accountability: theoretical guarantees (mathematical proofs) or statistical evidence can also serve as guarantees [**?**], however, in terms of explanations, what form they take and how they are proven correct are still open questions [**?**].

2.2.2 Explanation and Communication

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

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

2.2.3 Mechanics of Model Interpretation

How do we interpret models? Methods for developing interpretation models include: decision trees [**? ? ? ? ?**], decision tables [**? ?**] and decision sets [**? ?**]; input gradients, gradient vectors or sensitivity analysis [**? ? ? ? ?**]; exemplars [**? ?**];

[4] `https://www.eugdpr.org` last accessed 13 August 2018.

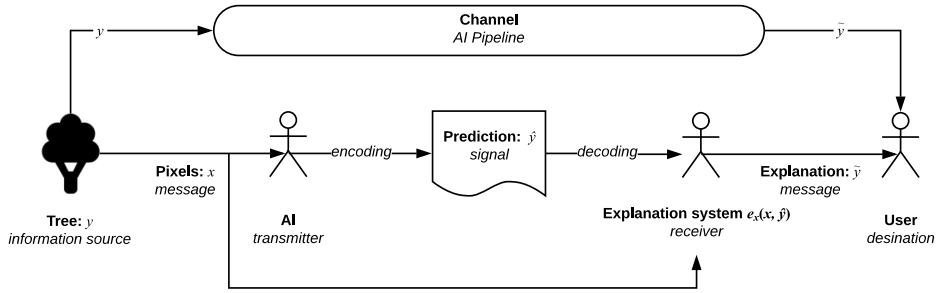


Figure 2.6: Theory of AI communication from information source, y, to intended user as explanations $\tilde{y}$.

generalised additive models [**?**]; classification (*if-then*) rules [**?** **?** **?** **?** **?**] and falling rule lists [**?**]; nearest neighbours [**?** **?** **?** **?** **?**] and Naïve Bayes analysis [**?** **?** **?** **?** **?** **?** **?** **?**].

Cross-domain studies have assessed the interpretability of these techniques against end-users, measuring response time, accuracy in model response and user confidence [**?** **?** **?** **?** **?** **?** **?** **?**], although it is generally agreed that decision rules and decision tables provide the most interpretation in non-linear models such as SVMs or NNs [**?** **?** **?**]. For an extensive survey of the benefits and fallbacks of these techniques, we refer to **?**], **?**] and **?**].

2.3 Application Programming Interfaces

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

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

While there are different types of APIs, such as software library/framework APIs for building desktop software, operating system APIs for interacting with the operating system, remote APIs for communication of varying technologies through common protocols, we focus on web APIs for communication of resources over the web (being the common architecture of cloud-based services). Further information on the development, usage and documentation of web APIs is provided in **??**.

2.3.1 API Usability

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

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

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

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

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

A later study by **?**] assessed the importance of a programmer's conceptual understanding of the background behind the task before implementing the task itself, a notion that we find most relevant for users of IWS APIs. While the study did not focus on developing web APIs (rather implementing a Bluetooth application using platform-agnostic terminology), the study demonstrated how developers show little confidence in their own metacognitive judgements to understand and assess the feasibility of the intent of the API and understand the vocabulary and concepts within the domain (i.e., wireless connectivity). This indecision over what search results

were relevant in their searches ultimately hindered their progress implementing the functionality, again decreasing productivity. **?** suggest to improve API usability by introducing the background of the API and its relevant concepts using glossaries linked to tutorials to each of the major concepts, and then relate it back to how to implement the particular functionality.

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

CHAPTER 3

Research Methodology

Investigating software engineering practices is often a complex task as it is imperative to understand the social and cognitive processes around software engineers and not just the tools and processes used [**?**]. This chapter explores our research methodology by exploring five key elements of empirical software engineering research: firstly, (i) we provide an extended focus to the study by reviewing our research questions (see **??**) anchored under the context of an existing classification taxonomy, (ii) characterise our research goals through an explicit philosophical stance, (iii) explain how the stance selected impacts our selection of research methods and data collection techniques (by dissecting our choice of methods used to reach these research goals), (iv) discuss a set of criteria for assessing the validity of our study design and the findings of our research, and lastly (v) discuss the practical considerations of our chosen methods.

The foundations for developing this research methodology has been expanded from that proposed by **?**], **?**], **?**] and **?**].

3.1 Research Questions Revisited

In **??**, we introduce three hypothesis of this study (RH1–RH3), namely: (i) existing IWS APIs are poorly documented for general use (RH1); (ii) existing IWS APIs do not provide sufficient metadata when used in context-specific use cases (RH2); and (iii) the combination of improving documentation and metadata will ultimately improve one of software quality, developer productivity and/or developer understanding (RH3).

To discuss our research strategy, we revisit our research questions through the classification technique discussed by **?**], a technique originally proposed in the field of psychology by **?**] but adapted to software engineering. Our research study involves a mix of five *knowledge questions*, that focus on existing practices and the ways in which they work, and two *design questions*, that focuses on designing better

ways to approach software engineering tasks [**?**]. Both classes of questions are respectively concerned with empirical and non-empirical software engineering that, in practice, are best combined in long-term software engineering research studies (such as this one) as they assist in tackling the investigation of a specific problem, approaches to solve that problem and finding what solutions work best [**?**].

3.1.1 Knowledge Questions

In total, five knowledge questions are posed in this study to help us understand the way developers currently interact and work with an IWS API; two exploratory, one base-rate, and two relationship and causality questions.

We begin by formulating two *exploratory questions* to attempt to better understand the phenomena of poor API documentation and metadata; both **??** and **??** respectively describe and classify what practices are in use for existing IWS API documentation and what problems currently exist when no metadata is returned. Answering these two questions assists in refining preciser terms of the phenomena, ways in which we find evidence for them and ensuring the data found is valid.

By answering these questions, we have a clearer understanding of the phenomena; we then follow up by posing an additional *base-rate question* that helps provide a basis to confirm that the phenomena occurring is normal (or unusual) behaviour by investigating the patterns of phenomena's occurrence. **??** is a descriptive-process question to help us understand how the developer currently understands existing IWS API documentation, given their lack of formal extended training in artificial intelligence. This achieves us an insight into the developer's mindset and regular thought patterns toward these APIs.

Lastly, we investigate the relationship between the improved documentation and improvements to other aspects of the software development process. Chiefly, **??** is concerned with whether any improvements to metadata or documentation correlate to improvements in software quality, developer productivity, or developer education (and is a *relationship establishment question*). If we establish such a relationship, we refine the question and investigate the specific causes using three *causality questions* defined under **??**, namely by associating three classes of measurable metrics (internal quality metrics, external quality metrics, developer education insight metrics) to the improved documentation.

3.1.2 Design Questions

?? and **??** are both *design questions*; they are concerned with ways in which we can improve an IWS by investigating what additional attributes are needed in both the documentation and metadata that assist developers to achieve their goals. They are not classified as knowledge questions as we investigate what *will be* and not *what is*. By understanding the process by which developers desire additional attributes of metadata and documentation, we can help shape improvements to the existing design of an IWS.

3.2 Philosophical Stances

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

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

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

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

Critical Theory This study chiefly adopts the philosophy of critical theory [**?**]. A key outcome of the study is to shift the developer's restrictive deterministic mindset and shed light on developing a new framework actively with the developer community that seeks to improve the process of using such APIs. In software engineering, critical theory is used to "actively [seek] to challenge existing perceptions about software practice" [**?**], and this study utilises such an approach to shift the mindset of IWS consumers and providers alike on how the documentation and metadata should not be written with the 'traditional' deterministic mindset at heart. Thus, our key philosophical approach is critical theory to seek out *what-can-be* using partial constructivism to model the current *what-is*.

3.3 Research Design

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

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

3.3.1 Review of Relevant Research Methods

Below we review some of the research methods most relevant to our research questions as refined in **??** as presented by **?**].

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

While we have defined hypotheses (RH1–RH3), refining them into precise, measurable variables is challenging due to the qualitative nature they present. A well-defined population is also critical and must be easily accessible; the varied range of beginner to expert software engineers with varied understanding of artificial intelligence concepts is required to perform controlled experiments, and thus recruitment may prove challenging. Lastly, the controlled experiment is essentially reductionist by affecting a small amount of variables of interest and controlling all others. This approach is too clinical for the practical outcomes by which our research goals aim for, and is therefore closely tied to the positivist stance.

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

Case studies are well-suited where the context of a situation plays a role in the phenomenon being studied, which we specifically relate back to RH2 (**??** and **??**) in exploring whether the context of an application using an IWS requires the IWS context-specific or of context-agnostic. They also lend themselves to purposive sampling rather than random sampling, and thus we can selective choose cases that benefit the research goal of RH2 and (using our critical theorist stance) select cases that will actively benefit our participant software engineering audience most to draw attention to situations regarded as most problematic.

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

By identifying representative sample of the population, from beginner to experienced developers with varying understanding of IWS APIs, we can use survey research to assist in answering our exploratory and base-rate research questions under RH1 and RH2 (see **??**) in determining the qualitative aspects of how individual developers perceive and work with the existing APIs, either by directly asking them or by mining third-party discussion websites such as Stack Overflow (SO). However, with direct survey research techniques, low response rates may prove challenging, especially if no inducements can be offered for participation.

Ethnographies Ethnographies investigates the understanding of social interaction within community through field observation [**?**]. Resulting ethnographies help understand how software engineering technical communities build practices, communication strategies and perform technical work collaboratively.

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

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

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

3.3.2 Review of Data Collection Techniques for Field Studies

? developed a taxonomy [**? ?**] showcasing data collection techniques in field studies that are used in conjunction with a variety of methods based on the level of interaction between researcher and software engineer, if any. This taxonomy is reproduced in **??**.

3.4 Proposed Experiments

This section discusses two proposed experiments that we conduct in this study. For each experiment, we describe an overview of the experiment grounded known methods and techniques (**????**), our approach to analysing the data, as well as linking the experiment back to a research hypothesis and question (**??**). A high-level overview of the proposed experiments and the major bodies of work they encompass is presented in **??**.

3.4.1 Experiment I: Develop Initial Framework

Experiment I shapes a context-agnostic approach to understand current usage patterns of IWS APIs and the ways by which developers interpret them. Briefly, this experiment is comprised under two phases of field survey research: (i) repository mining developer discussion forums (i.e., analysis of databases and documentation analysis) to understand what developers currently complain about on these forums and where their mismatch in understanding lies; (ii) conducting unstructured interviews and distributing a questionnaire to gather personal opinion based on individual developer's anecdotal remarks.

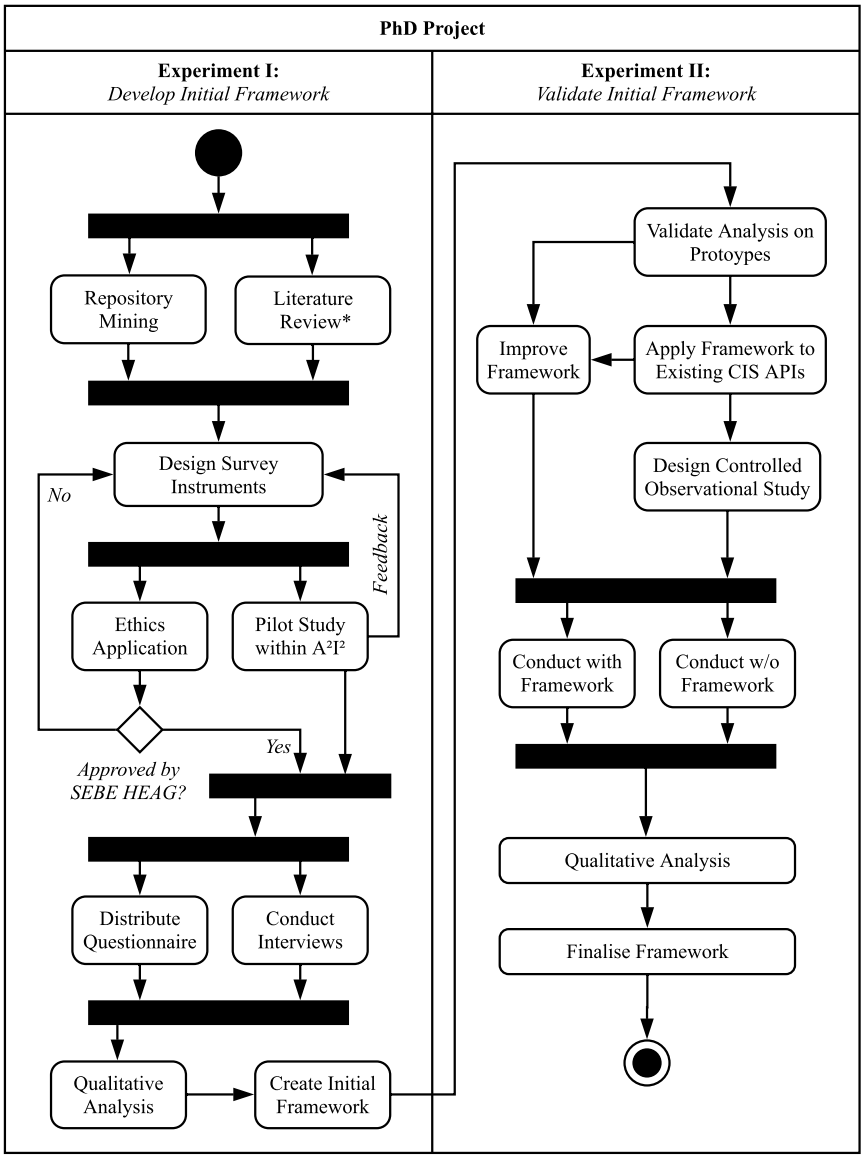


Figure 3.1: High-level activity diagram of the proposed experiments in this study. Literature review is ongoing.

Relevance and Motivation

Experiment I aims to better understand the existing mindsets that developers have when approaching to use computer vision services (CVSs). This work therefore ties in to RH1; by understanding the developer mindset in how they interpret CVS APIs, we are better informed to produce a framework that increases the effectiveness of the documentation of those existing CVS providers.

RH1 postulates that the software engineering community do not fully understand the 'magic' behind IWS APIs. As described in **??**, they face a gap in their

understanding around the underlying architecture of pre-built, machine learnt APIs (**??**). Software developers are not well-supported by the IWS providers, and therefore do not have a consistent set of common best practices when approaching to use these APIs (**??**). It is therefore necessary that IWS providers provide additional information to gap this mismatched understanding (**??**).

Data Collection & Analysis

Phase 1: Repository Mining 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 [**?**].

Thus, to begin validating IWS API usage and misunderstanding in a generalised context (i.e., context-agnostic to the project at hand), we propose using repository mining on SO to help answer our research questions. Specifically, we select SO due to its targeted community of developers[1] and the availability of its publicly available dataset released as 'data dumps' on the Stack Exchange Data Explorer[2] and Google BigQuery[3]. Studies conducted have also used SO to mine developer discourse [**? ? ? ? ? ? ? ? ? ? ? ?**].

Due to the enormity of the data produced, we will use qualitative analysis on the questions mined using assistive tools such as NVivo. For this, we will conduct a thematic analysis on the themes of each question mined, the relevance of the question to our research topic, and ensuring strict coding schemes (that reflect our research goals) are adhered to. We refer to **?**] and **?**] on coding and analysing this qualitative data gathered.

Phase 2: Personal Opinion Surveys We follow the triangulation approach proposed by **?**] to corroborate the qualitative data of developers' discussion of SO with secondary survey research, thereby validating what people say on git with what is said and done in real life. **?**] provide an introduction on methods used to conduct personal opinion surveys which we adopt 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 two survey instruments (consisting of a structured questionnaire and semi-structured interview), (iv) evaluating our instruments, (v) obtaining the data and (vi) analysing the data.

[1] We also acknowledge that there are other targeted software engineering Stack Exchange communities such as Stack Exchange Software Engineering (`https://softwareengineering.stackexchange.com`), though (as of January 2019) this much smaller community consists of only 52,000 questions versus SO's 17 million.

[2] `https://data.stackexchange.com/stackoverflow` last accessed 17 January 2017.

[3] `https://console.cloud.google.com/marketplace/details/stack-exchange/stack-overflow` last accessed 17 January 2017.

As is good practice in developing questionnaire instruments to evaluate their reliability and validity [**?**], we evaluate our instrument design by asking colleagues to critique it via pilot studies within A^2I^2. This assists in identifying any problems with the questionnaire itself and with any issues that may occur with the response rate and follow-up procedures. We follow a similar approach by practicing the interview instrument on colleagues within A^2I^2.

Findings from the pilot study helps inform us for a widely distributed questionnaire and conducting interviews out in the field, where we recruit external software engineers in industry through the industry contacts of A^2I^2. Ethics approval from the Faculty of Science, Engineering and Built Environment Human Ethics Advisory Group (SEBE HEAG) will be required prior to externally conducting this survey research (see **??**). The quantitative (survey) and qualitative (interview) analysis allows us to shape the research outcome of RH1—an API documentation quality assessment framework—and assists in stabilising our general understanding of how developers use these existing APIs.

3.4.2 Developing the Initial Framework

Our initial framework is developed using the qualitative and quantitative analyses from the findings of Experiment I. As this is a creative phase in which we are developing a new framework, the exact process by which we develop the initial framework will come to light once more insight is determined. However it is anticipated discussion with other researchers and engineers at A^2I^2 about the analyses of the findings (i.e., white-boarding sessions of potential ideas from the findings) will help develop our initial documentation framework. This framework will take the shape of a checklist or table, typical of information systems studies (e.g., [**?**]), that indicate what attributes should be best suited for what needs.

3.4.3 Experiment II: Validate Initial Framework

Experiment II extends the *generalised* context of Experiment I by evaluating how the findings of Experiment I translates to context-specific applications. We confirm that the generalised findings are (indeed) genuine by conducting action research in combination with an observational study on software engineers. This experiment is also compromised of: (i) development of prototypes using IWS APIs of differing contexts; (ii) presenting a solution framework to developers to interpret the improvement of their understanding when using an IWS.

Relevance and Motivation

Experiment II aims to contextualise the findings from Experiment I; that is, if we add *varying contexts* to the applications we write using IWS APIs, what is needed to extend the *context-agnostic* framework developed in Experiment I? This work relates back to RH2; adding context-specific metadata to the endpoints of these APIs, we can highlight what issues exist when such metadata is not present (**??**) and what types of metadata developers seek (**??**).

Moreover, the implication of the first two hypotheses suggest that applying an API documentation and metadata quality assessment framework may have an effect on other aspects within the software engineering process (RH3). Thus, this experiment also confirms if our framework makes an improvement to software quality, developer productivity and/or developer informativeness (**??** and **??**).

Data Collection & Analysis

To confirm findings of the method within RH1 is genuine, we shift from reviewing the documentation from a general stance to a specialised (context-specific) stance in the use of these APIs.

This is firstly achieved by using existing IWS APIs to develop basic 'prototypes', each having differing contexts. The number of prototypes to develop and the use cases they have will be informed by the results of Experiment I, and therefore cannot yet be described at this stage. Our action research in developing the prototypes will help inform any potential gaps that exist in the findings of RH1, especially with regards to context-specifity, and therefore improves the metadata component of our framework (as per the outcome of RH2).

This outcome will also help us design the next stage of the experiment, consisting of a comparative controlled study [**?**] to capture firsthand behaviours and interactions toward how software engineers approach using an IWS with and without our framework applied. We will provide improved documentation and metadata responses of a set of popular CVSs that is documented with the additional metadata and whose information is organised using our framework.

We then recruit 20 developers of varying experience (from beginner programmer to principal engineer) to complete five tasks under an observational, comparative controlled study, 10 of which will (a) develop with the *new* framework, and the other 10 will (b) develop with the *as-is/existing* documentation. From this, we compare if the framework makes improvements by capturing metrics and recording the observational sessions for qualitative analysis. We use visual modelling to analyse the qualitative data using matrices [**?**], maps and networks [**?**] as these help illustrate any causal, temporal or contextual relationships that may exist to map out the developer's mindset and difference in approaching the two sets of designs of the same tasks.

3.5 Empirical Validity

In **??**, we state that this study primarily adopts a critical theorist stance. Critical theorists assess research quality by the utility of the knowledge gained [**?**]. **?**] established criteria on validating information systems research specifically for action research unifying four dimensions of the research (conceptual foundation, study design, research process, and role expectations) against 22 varying criteria. We also partially adopt constructivism as we attempt to model the developer mindset rich in qualitative data, to which eight strategies identified by **?**] cover.

We identify possible threats to internal- (study design), external- (generality of results), and construct-validity (theoretical understanding) in the following sections, and describe how we mitigate these threats.

3.5.1 Threats to Internal Validity

Hawthorne Effect

Observational field techniques involving participants often run a risk producing misaligned results from laboratory versus environmental (practical) conditions. This is commonly known as the Hawthorne effect [**?** **?**] and careful consideration of this effect must be reflected when designing our controlled observation (**??**). We aim to carefully explain the purpose and protocol to research participants, encouraging them to act as much as possible as in their practical conditions. We also encourage the 'think-aloud' to participants protocol to reinforce this. By highlighting the Hawthorne effect to them, we anticipate that participants will be aware of the condition, and therefore avoid doing things that do not reflect real-world action.

Misleading Statements in Interviews

Similarly, threats to the interview survey instrument exist where participants do not often report differences in behaviour from what they actually do in practice [**?**]. We anticipate that conducting interviews in a semi-structured manner may assist in following up with unexpected statements (as opposed to structured interviews) and additionally corroborate findings using **?** 's concurrent triangulation method [**?**] to verify potentially misleading statements from participants with questionnaire results and observation findings.

Participant Observation Accuracy

Conducting participant observations is a skill that requires training. While every effort will be instilled to ensure all relevant observations are noted, it is impossible for a single observer to note every possible interaction that occurs in all observations made. Therefore, to validate the consistency of data collected, we may require rater agreement exercises [**?**] and we will likely use a form of recording device (with participant consent) to ensure all information is transcribed correctly in the interview.

Unintended Interviewee Bias

Interviewers should introduce the research by which participants are involved in by describing an expiation of the research being conducted. However, the amount of information described may impact the bias instilled on the interviewee. For example, if the participant does not understand the goals of the study or feel that they are of the 'right target', then it is likely that they may choose not to be involved in the study or give misleading answers. On the other hand, if interviewees are told too much information, then they may filter responses and leave out vital data that the

interviewer may be interested in. To mitigate this, varying levels of information will be 'tested' against colleagues to determine the right level of how much information is divulged at the beginning of the interview.

Poor Questionnaire Responses

Unless significant inducements are offered, **?**] report that a consistent response rate of 5% has been found in software engineering questionnaires distributed and in information systems the median response rates for surveys are 60% [**?**]. We observe that low response rates may adversely effect the findings of our survey, typically as software engineers find little time to do them. Tackling this issue can be resolved by carefully designing succinct, unambiguous and well-worded questions that we will verify against our colleagues and within the pilot study in A^2I^2, wherein any adjustments made from the pilot study due to unexpected poor quality of the questionnaire will be reported and explained. We also adopt research conducted in the field of questionnaire design, such as ensuring all scales are worded with labels [**?**] or using a summating rating scale [**?**] to address a specific topic of interest if people are to make mistakes in their response or answer in different ways at different times. Similar effects exist to that above where what occurs in reality is not what is reflected in our results; we refer to our concurrent triangulation approach to gap this risk.

3.5.2 Threats to External Validity

Representative Sample Size

Our results must generalise by ensuring a representative subset of the target population is found. If results do not generalise, then all that is found is potentially of little more value than personal anecdote [**?**]. Therefore, designing a well-defined sample frame to determine which developers we wish to target is empirical. For this, we refer to **?**].

Student Cohorts

External validity is typically undermined when students are recruited in software engineering research, which is common practice [**?**]. Analytical argument is required to describe why results on students are reflective of results found on developers in industry. Therefore, we anticipate that—through industry contacts at A^2I^2—we will be able to contact developers in industry, thereby minimising our reliance to use students as participants.

Concurrent Triangulation Strategy

A drawback with the concurrent triangulation strategy is that multiple sources of data are concurrently collected within the same time. Collecting and analysing data *sequentially*, instead of concurrently, allows for time to analyse data between studies, thus allowing each analysis to adapt as more emerging results are explored. **?**]

states that the challenge in this approach is that it may be difficult for researchers to compare results of multiple analyses or resolve contradictions that begin to arise when this is performed concurrently. A mitigation strategy, should this occur, would be to seek out further sources of evidence, or even re-conduct a follow-up study if time permits.

3.5.3 Threats to Construct Validity

Developer Informativeness

RH3 describes that if we improve the documentation of IWS APIs, then developers are more informed/educated in what they do. This therefore increases their productivity and the quality of the applications they build. However, the construct of 'informativeness' is difficult to capture with standalone metrics, and using simple quantitative metrics such as time taken to complete a task or lines of code to implement it may not reflect that a developer is more 'informed'. Therefore, we propose further investigation into understanding how to measure informativeness of software engineers to ensure that this construct validity does not impact our results too greatly.

Part II

Publications

CHAPTER 4

Identifying Evolution in Computer Vision Services[†]

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

4.1 Introduction

The availability of intelligent web services (IWSs) has made artificial intelligence (AI) tooling accessible to software developers and promises a lower entry barrier for their utilisation. Consider state-of-the-art computer vision (CV) analysers, which require either manually training a deep-learning classifier, or selecting a pre-trained model and deploying these into an appropriate infrastructure. Either are laborious in time, and require non-trivial expertise along with a large data set when training or customisation is needed. In contrast, IWSs providing CV (i.e., computer vision services or CVSs such as [**? ? ? ? ? ? ? ? ? ? ? ? ? ? ?**]) abstract these complexities behind a web application programming interface (API) call. This removes the need to understand the complexities required of machine learning (ML), and requires little more than the knowledge on how to use RESTful endpoints. The

[†]This chapter is originally based on . Terminology has been updated to fit this thesis.

ubiquity of these services is exemplified through their rapid uptake in applications such as aiding the vision-impaired [**? ?**].

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

ML typically involves use of statistical techniques that yield components with a non-deterministic external behaviour; that is, for the same given input, different outcomes may result. However, developers, in general, are used to libraries and small components behaving predictably, while systems that rely on ML techniques work on confidence intervals[1] and probabilities. For example, the developer's mindset suggests that an image of a border collie—if sent to three intelligent computer vision services (CVSs)—would return the label 'dog' consistently with time regardless of which service is used. However, one service may yield the specific dog breed, 'border collie', another service may yield a permutation of that breed, 'collie', and another may yield broader results, such as 'animal'; each with results of varying confidence values.[2] Furthermore, the third service may evolve with time, and thus learn that the 'animal' is actually a 'dog' or even a 'collie'. The outcomes are thus behaviourally inconsistent between services providing conceptually similar functionality. As a thought exercise, consider if the sub-string function were created using ML techniques—it would perform its operation with a confidence where the expected outcome and the AI inferred output match as a *probability*, rather than a deterministic (constant) outcome. How would this affect the developers' approach to using such a function? Would they actively take into consideration the non-deterministic nature of the result?

Myriad software quality models and software engineering (SE) practices advocate maintainability and reliability as primary characteristics; stability, testability, fault tolerance, changeability and maturity are all concerns for quality in software components [**? ? ?**] and one must factor these in with consideration to software evolution challenges [**? ? ? ? ?**]. However, the effect this non-deterministic behaviour has on quality when masked behind an IWS is still under-explored to date in SE literature, to our knowledge. Where software depends on IWSs to achieve functionality, these quality characteristics may not be achieved, and developers need to be wary of the unintended side effects and inconsistency that exists when using non-deterministic components. A CVS may encapsulate deep-learning strategies or stochastic methods to perform image analysis, but developers are more likely to approach IWSs with a mindset that anticipates consistency. Although the documentation does hint at this non-deterministic behaviour (i.e., the descriptions of 'confidence' in various CVSs suggest the they are not always confident, and thus not

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

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

deterministic [**? ? ?**]), the integration mechanisms offered by popular vendors do not seem to fully expose the nuances, and developers are not yet familiar with the trade-offs.

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

4.2 Motivating Example

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

Rather than building a CV engine from scratch, Rosa thinks she can achieve this using one of the popular CVSs (e.g., [**? ? ? ? ? ? ? ? ? ? ? ? ? ? ?**]). However, Rosa comes from a typical software engineering background with limited knowledge of the underlying deep-learning techniques and implementations as currently used in CV. Not unexpectedly, she internalises a mindset of how such services work and behave based on her experience of using software libraries offered by various SDKs. This mindset assumes that different cloud vendor image processing APIs more-or-less provide similar functionality, with only minor variations. For example, cloud object storage for Amazon S3 is both conceptually and behaviourally very similar to that of Google Cloud Storage or Azure Storage. Rosa assumes the CVSs of these platforms will, therefore, likely be very similar. Similarly, consider the string libraries Rosa will use for the app. The conceptual and behavioural similarities are consistent; a string library in Java (Android) is conceptually very similar to the string library she will use in Swift (iOS), and likewise both behave similarly by providing the same results for their respective sub-string functionality. However, **unlike the cloud storage and string libraries, different CVSs often present conceptually similar functionality but are behaviourally very different**. IWS vendors also hide the depth of knowledge needed to use these effectively—for instance, the training data set and ontologies used to create these services are hidden in the documentation. Thus, Rosa isn't even exposed to this knowledge as she reads through the documentation of the providers and, thus, Rosa makes the following assumptions:

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

4.3 Related Work

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

4.3.1 External Quality

Robustness for safety-critical applications A typical focus of recent work has been to investigate the robustness of deep-learning within CV technique implementation, thereby informing the effectiveness in the context of the end-product. The common method for this has been via the use of adversarial examples [**?**], where input images are slightly perturbed to maximise prediction error but are still interpretable to humans.

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

The results suggest that current state-of-the-art CV techniques may not be robust enough for safety critical applications as they do not handle intentional or unintentional adversarial attacks. Moreover, as such adversarial examples exist in the physical world [**?** **?**], "the natural world may be adversarial enough" [**?**] to fool AI software. Though some limitations and guidelines have been explored in this

area, the perspective of *Intelligent* Web Services is yet to be considered and specific guidelines do not yet exist when using CVSs.

Testing strategies in ML applications Although much work applies ML techniques to automate testing strategies, there is only a growing emphasis that considers this in the opposite sense; that is, testing to ensure the ML product works correctly. There are few reliable test oracles that ensure if an ML has been implemented to serve its algorithm and use case purposefully; indeed, "the non-deterministic nature of many training algorithms makes testing of models even more challenging" [**?**]. **?**] proposed a SE-based testing approach on ML ranking algorithms to evaluate the 'correctness' of the implementation on a real-world data set and problem domain, whereby discrepancies were found from the formal mathematical proofs of the ML algorithm and the implementation.

Recently, **?**] conducted a comprehensive review of testing strategies in ML software, proposing several research directions and recommendations in how best to apply SE testing practices in ML programs. However, much of the area of this work specifically targets ML engineers, and not application developers. Little has been investigated on how application developers perceive and understand ML concepts, given a lack of formal training; we note that other testing strategies and frameworks proposed (e.g., [**?** **?** **?**]) are targeted chiefly to the ML engineer, and not the application developer.

However, **?**] recently demonstrated (using real-world ML projects) the developmental challenges posed to developers, particularly those that arise when there is a lack of transparency on the models used and how to troubleshoot ML frameworks using traditional SE debugging tools. This said, there is no further investigations into challenges when using the higher, 'ML friendly' layers (e.g., IWSs) of the 'machine learning spectrum' [**?**], rather than the 'lower layers' consisting of existing ML frameworks and algorithms targeted toward the ML community.

4.3.2 Internal Quality

Quality metrics for cloud services CVSs are based on cloud computing fundamentals under a subset of the Platform as a Service (PaaS) model. There has been work in the evaluation of PaaS in terms of quality attributes [**?**]: these attributes are exposed using service-level agreements (SLAs) between vendors and customers, and customers denote their demanded quality of service (QoS) to ensure the cloud services adhere to measurable KPI attributes.

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

specific to the ML service may have significant legal ramifications.

Web service documentation and documenting ML From the *developer's* perspective, little has been achieved to assess IWS quality or assure quality of these CVSs. web service and their APIs are the bridge between developers' needs and the software components [**?**]; therefore, assessing such CVSs from the quality of their APIs is thereby directly related to the development quality [**?**]. Good APIs should be intuitive and require less documentation browsing [**?**], thereby increasing productivity. Conversely, poor APIs that are hard to understand and work with reduce developer productivity, thereby reducing product quality. This typically leads to developers congregating on forums such as Stack Overflow, leading to a repository of unstructured knowledge likely to concern API design [**?**]. The consequences of addressing these concerns in development leads to a higher demand in technical support (as measured in [**?**]) that, ultimately, causes the maintenance to be far more expensive, a phenomenon widely known in software engineering economics [**?**]. Rosa, for instance, isn't aware of technical ML concepts; if she cannot reason about what search results are relevant when browsing the service and understanding functionality, her productivity is significantly decreased. Conceptual understanding is critical for using APIs, as demonstrated by **?** , and the effects of maintenance this may have in the future of her application is unknown.

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

However, while target audiences for these documents may be of a more technical AI level (i.e., the ML engineer), there is still no standardised communication format for application developers to reason about using particular IWSs, and the ramifications this may have on the applications they write is not fully conveyed. Hence, our 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 [**?**], Google Cloud Vision [**?**], Azure Computer Vision [**?**]. Specifically, we targeted the AWS `DetectLabels` endpoint [**?**], the Google Cloud Vision `annotate:images` endpoint [**?**] and Azure's `analyze` endpoint [**?**]. 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 [**?**] 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 [**? ? ? ? ? ? ? **] also exist, including other popular services used in Asia [**? ? ? ? **] (some offering 3D image analysis [**?**]). We reflect on the impacts this has to our study design in **??**.

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 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 [**?**]. We have made our other two data sets available online ([**?**]). We collected results and their responses from each service's API endpoint using a python script [**?**] that sent requests to each service periodically via cron jobs. **??** 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

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

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.

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

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

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 ∩ B ∩ C	3.33%	2.73%	4.68%	2.75%	0.0100
A ∩ B	6.67%	11.27%	12.26%	10.07%	0.0299
A ∩ C	20.00%	13.94%	17.28%	17.07%	0.0304
B ∩ C	6.67%	12.97%	20.90%	13.51%	0.0713

4.5 Findings

4.5.1 Consistency of top labels

Consistency across services **??** presents the consistency of the top labels between data sets, as measured by the cardinality of the intersection of all three services’ set of top labels divided by the number of images per data set. A combination of services present varied overlaps in their top labels; services A and C provide the best overlap for all three data sets, however the intersection of all three irrespective of data sets is low.

The implication here is that, without semantic comparison (see **??**), 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 **??** 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 **??**’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 **??** shows no intersection at all. Furthermore, the part of the image each service focuses on may not be consistent for their top labels; in **??**, 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

(a)

(b)

Figure 4.2: *Left:* The top labels for each service do not intersect, with each having a varied ontology: T_i = { A = {'black'}, B = {'indoor'}, C = {'slide', '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 = { 'carrot' }, B = { 'indoor' }, C = { 'spoon' } }.

(a)

(b)

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.

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 **??** 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, CVSs do not agree on the top label for images, and therefore developers are likely to be vendor locked, making changes between services non-trivial.

Semantic consistency where $|T_i| > 1$ Service C returns two top labels for **??**; 'slide' and 'toy'. More than one top label is typically returned in service C (80.00%,

56.97%, and 81.66% of all images for all three data sets, respectively) though this also occurs in B in the large (4.97% of all images) and COCOVal17 data sets (2.38%). Semantic inconsistencies of what this label conceptually represents becomes a concern as these labels have confidences of *equal highest* consistency. Thus, some services are inconsistent in themselves and cannot give a guaranteed answer of what exists in an image; services C and B have multiple top labels, but the respective services cannot 'agree' on what the top label actually is. In **??**, service C presents a reasonably high confidence for the set of 7 top labels it returns, however there is too much diversity ranging from a 'hot chocolate' to the hypernym 'food'. Both are technically correct, but it is up to the developer to decide the level of hypernymy to label the image as. We also observe a similar effect in **??**, where the image is labelled with both the subject matter and the number of subjects per image.

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

> *"Identify and categorize an entire image, using a category taxonomy with parent/child hereditary hierarchies. Categories can be used alone, or with our new tagging models."* [**?**]

Thus, even if Rosa implemented conceptual similarity analysis for the image, the top label set may not provide sufficient information to derive at a conclusive answer, and if simply relying on only one label in this set, information such as the duplicity of objects (e.g., 'several' in **??**) may be missed.

4.5.2 Consistency of confidence

Consistency of top label's confidence In **??**, 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.

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 **??**, the three services agree with $\sigma = 0.277$, significantly larger than that of all images in general

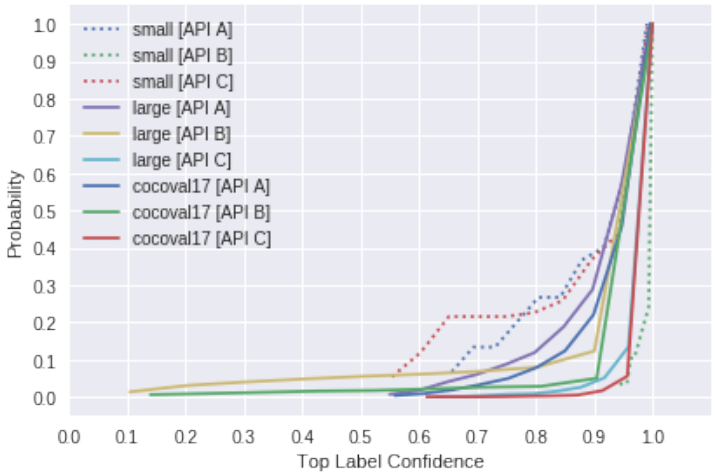


Figure 4.4: Cumulative distribution of the top labels' confidences. One in nine images return a top label(s) confident to $\gtrapprox 97\%$, though there is a wider distribution for service A.

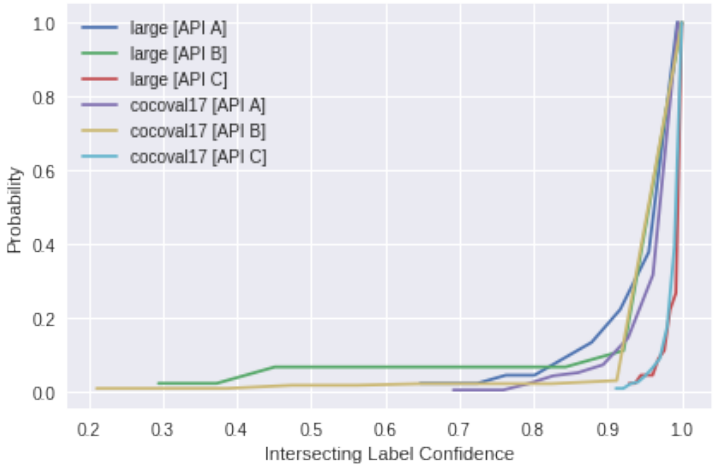


Figure 4.5: Cumulative distribution of intersecting top labels' confidences. The small data set is intentionally removed due to low intersections of labels (see **??**).

Table 4.3: 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]

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%.

$\sigma = 0.0831$. **??** displays the cumulative distribution of all intersecting top labels' confidence values, presenting slightly similar results to that of **??**.

4.5.3 Evolution risk

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 COCO-Val17 data sets in service A. Thus, top labels are stable but not guaranteed to be constant.

Confidence Stability Similarly, where the top label(s) remained the same from one interval to the next, the confidence values were stable. **??** displays the proportion of images that changed their top label's confidence values with various statistics on the confidence deltas between snapshots (δ_c). However, this delta is so minuscule that we attribute such changes to statistical noise.

4.6 Recommendations

4.6.1 Recommendations for IWS users

Test with a representative ontology for the particular use case Rosa should ensure that in her testing strategies for the app she develops, there is an ontology focus for the types of vocabulary that are returned. Additionally, we noted that there

was a sudden change in case for services A and C; for all comparative purposes of labels, each label should be lower-cased.

Incorporate a specialised IWS testing methodology into the development lifecycle Rosa can utilise the different aspects of consistency as outlined in this paper as part of her quality strategy. To ensure results are correct over time, we recommend developers create a representative data set of the intended application's data set and evaluate these changes against their chosen service frequently. This will help identify when changes, if any, have occurred if vendors do not provide a line of communication when this occurs.

IWSs are not 'plug-and-play' Rosa will be locked into whichever vendor she chooses as there is inherent inconsistency between these services in both the vocabulary and ontologies that they use. We have demonstrated that very few services overlap in their vocabularies, chiefly because they are still in early development and there is yet to be an established, standardised vocabulary that can be shared amongst the different vendors. Issues such as those shown in **??** can therefore be avoided.

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

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

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

4.6.2 Recommendations for IWS providers

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

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.

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 [**? ?**] 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.

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 CV techniques may have limits in what they can and cannot detect, but this limitation can be exposed in the documentation to the developers.

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

4.7 Threats to Validity

4.7.1 Internal Validity

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

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

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

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

4.7.2 External Validity

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

Moreover, in this paper we have investigated *computer vision* services. Thus, the significance of our results to other domains such as natural language processing or audio transcription is, therefore, unknown. Future studies may wish to repeat our methodology on other domains to validate if similar patterns occur; we remain this open for future work.

4.7.3 Construct Validity

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

4.8 Conclusions & Future Work

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

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

There are many potential directions stemming from this work. To start, we plan to focus on preparing a more comprehensive datasheet specifically targeted at what should be documented to application developers, and not data scientists. Reapplying

this work in real-world contexts, that is, to get real developer opinions and study production grade systems, would also be beneficial to understand these phenomena in-context. This will help us clarify if such changes are a real concern for developers (i.e., if they really need to change between services, or the service evolution has real impact on their applications). We also wish to refine and systematise the method used in this study and develop change detectors that can be used to identify evolution in these services that can be applied to specific ML domains (i.e., not just CV), data sets, and API endpoints, thereby assisting application developers in their testing strategies. Moreover, future studies may wish to expand the methodology applied by refining how the responses are compared. As there does not yet exist a standardised list of terms available between services, labels could be *semantically* compared instead of using exact matches (e.g., by using stem words and synonyms to compare similar meanings of these labels), similar to previous studies [**?**].

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

CHAPTER 5

Systematic Mapping Study of API Documentation Knowledge[†]

Abstract Good application programming interface (API) documentation facilities the development process, improving productivity and quality. While the topic of API documentation quality has been of interest for the last two decades, there have been few studies to map the specific constructs needed to create a good document. In effect, we still need a structured taxonomy against which to capture knowledge. This study reports emerging results of a systematic mapping study. We capture key conclusions from previous studies that assess API documentation quality, and synthesise the results into a single framework. By conducting a systematic review of 21 key works, we have developed a five dimensional taxonomy based on 34 categorised weighted recommendations. All studies utilise field study techniques to arrive at their recommendations, with seven studies employing some form of interview and questionnaire, and four conducting documentation analysis. The taxonomy we synthesise reinforces that usage description details (code snippets, tutorials, and reference documents) are generally highly weighted as helpful in API documentation, in addition to design rationale and presentation. We propose extensions to this study aligned to developer's utility for each of the taxonomy's categories.

5.1 Introduction

Improving the quality of application programming interface (API) documentation is highly valuable to the software development process; good documentation facilitates productivity and thus quality is better engineered into the system [**?**]. Where an application developer integrates new pieces of functionality (via APIs) into a system, their productivity is affected either by inadequate skills (*"I've never used an API like this, so must learn from scratch"*) or, where their skills are adequate, an imbalanced cognitive load that causes excessive context switching (*"I have the skills for this, but am confused or misunderstand"*). In the latter case, what causes this confusion

[†]This chapter is originally based on . Terminology has been updated to fit this thesis.

and how to mitigate it via improved API documentation is an area that has been explored; prior studies have provided recommendations based on both qualitative and quantitative analysis of developer's opinions. These recommendations and guidelines propose ways by which developers, managers and solution architects can construct systems better.

However, to date there has been little attempt to systematically capture this knowledge about API documentation from various studies into a readily accessible, consolidated format, that assists API designers to prepare better documentation. While previous works have covered certain aspects of API usage, many have lacked a systematic review of literature and do not offer a taxonomy to consolidate these guidelines together. For example, some studies have considered the technical implementation improving API usability or tools to generate (or validate) API documentation from its source code (e.g., [**?** **?** **?**]); there still lacks a consolidated effort to capture the knowledge and artefacts best suited to *manually write* API documentation.

⟨ TODO: *Introduce intelligent services documentation* ⟩ The need for these insights to be well-captured is evermore present with the introduction of intelligent web services (IWSs), 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 [**?**]. A popular subset of such components include computer vision services (CVSs), which use machine intelligence to make predictions on images such as object categorisation or facial recognition [**?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?** **?**]. The longer-term impacts of *poor documentation* for CVSs is largely under-explored and may have significant impacts of AI-first systems if the application developers who use them do not think in the non-deterministic mental model of the designers who create CVSs.

This paper presents outcomes from a preliminary work to address this gap and offers four key contributions:

- a systematic mapping study (SMS) consisting of 21 studies that capture what knowledge or artefacts should be contained within API documentation; and,
- a structured taxonomy based on the consolidated recommendations of these 21 studies.
- ⟨ TODO: *Relevance as per devs* ⟩ an assessment of the efficacy of these recommendations via a 'relevance ranking' for each of the 34 categories that empirically reflects what is important to document from a *practitioner* point of view;
- ⟨ TODO: *Assessment against docs* ⟩ a heuristic validation of the taxonomy against CVSs to assess where existing CVSs documentation needs improvement.

After performing our SMS on what API knowledge should be captured in documentation—to assist API designers—we propose a five dimensional taxonomy consisting of: (1) Usage Description; (2) Design Rationale; (3) Domain Concepts; (4) Support Artefacts; and (5) Documentation Presentation. ⟨ TODO: *Describe that this was weighted against devs opinions* ⟩ This taxonomy was then surveyed against

X developers to assess the relevance of these weightings from the practitioner's viewpoint. ⟨ TODO: *Describe our application of the taxonomy to CVS* ⟩ We then assess our taxonomy against a subset of IWSs—three popular CVS: Google Cloud Vision [**?**], AWS Rekognition [**?**] and Azure Computer Vision [**?**]. We anticipate that future API designers may use this resource as a means to improve the ways in which they communicate their mental models and patterns of thinking while developing these APIs.

This paper is structured as thus: **??** presents related work in the area; **??** is divided into two subsections, the first describing how primary sources were selected in a SMS with the second describing the development of our taxonomy from these sources; ⟨ TODO: *Introduce section where we validated study* ⟩ **??** describes how we adopted the System Usability Scale (SUS) to develop a survey instrument of 52 questions to validate the taxonomy and assess its efficacy against the three popular CVS selected; **??** presents the findings from our validation and the proposed taxonomy; **??** describes the threats to validity of this work and **??** provides concluding remarks and the future directions of this study.

5.2 Related Work

SMSs have previously been explored in the area of API usability and developer experience. **?**] recently performed a SMS on 36 API documentation generation tools and approaches. Presented is an analysis of state-of-the-art of the tools developed, what kind of documentation is generated by them, and the dependencies they require to generate this documentation. Their findings highlight a recent effort on the development of API documentation by producing example code snippets and/or templates on how to use the API or bootstrap developers to begin using the APIs. A secondary focus is closely followed by tools that produce natural language descriptions that can be produced within developer documentation. However, **?** produce a SMS on the types of *tooling* that exists to assist in producing and validating API documentation. While this is a systematic study with key insights into the types of tooling produced, there is still a gap for a SMS in what *guidelines* have been produced by the literature in developing natural-language documentation itself, which our work has addressed.

?] performed a heuristic assessment of 11 high-level universal design elements of API documentation against 35 popular APIs. He demonstrated that many of these popular APIs fail to grasp even the basic of these elements; for example, 25% of the documentation sets did not provide any basic overview documentation. However, the heuristic used within this study consists of just 11 elements and is based on only three seminal works. Our work extends these heuristics and structures them into a consolidated, hierarchical taxonomy using a systematic taxonomy development method for software engineering (SE).

A taxonomy of knowledge patterns within API reference documentation by **?**] classified 12 distinct knowledge types. Evaluation of the taxonomy against JDK 6 and .NET 4.0 showed that, while functionality and structure of the API is well-communicated, core concepts and rationale about the API are quite rare to find. Moreover, they demonstrated that low-value 'non-information' (documentation

that provides uninformative boilerplate text with no insight into the API at all) is substantially present in the documentation of methods and fields in these APIs. Their findings recommend that developers factor their 12 distinct knowledge types into the process of code documentation and prevent documenting low-value documentation. The development of their taxonomy consisted of questions to model knowledge and information, thereby capturing the reason about disparate information units independent to context; a key difference to this paper is the systematic taxonomy approach utilised.

⟨ TODO: *AC*: *Paragraph on the SUS*; *paragraph on ICVS* ⟩

5.3 Method

Our taxonomy development consisted of two phases. Firstly, we conducted an SMS to identify and analyse API documentation studies, following the guidelines of **?**] and **?**]. Following this, we followed the SE taxonomy development method devised by **?**] on our findings from the SMS.

5.3.1 Systematic Mapping Study

Research Questions (RQs) To guide our SMS, we developed the following RQs:

> **RQ1** What knowledge do API documentation studies contribute?
> **RQ2** How is API documentation studied?

The intent behind RQ1 is to collate as much of the insight provided by the literature on how API providers should best document their work. This helped us shape and form the taxonomy provided in **??**. RQ2 addresses methodologies by which these studies come to these conclusions to identify gaps in literature where future studies can potentially focus.

Automatic Filtering Informed by similar previous studies in SE [**?** **?** **?**], we begin by defining the SWEBOK [**?**] knowledge areas (KAs) to assist in the search and mapping process of an SMS. Our search query was built using related KAs, relevant synonyms, and the term 'software engineering' (for comprehensiveness) all joined with the OR operator. Due to the lack of a standard definition of an API, we include the terms: 'API' and its expanded term; software library, component and framework; and lastly SDK and its expanded term. These too were joined with the OR operator, appended with an AND. Lastly, the term 'documentation' was appended with an AND. Our final search string was:

Table 5.1: Summary of our 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

(“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)

The query was then executed on all available metadata (title, abstract and keywords) on three primary sources to search for relevant studies in May 2019. Web of Science[1] (WoS), Compendex/Inspec[2] (C/I) and Scopus[3] were chosen due to their relevance in SE literature (containing the IEEE, ACM, Springer and Elsevier databases) and their ability to support advanced queries [**?** **?**]. A total 4,501 results[4] were found, with 549 being duplicates. **??** displays our results in further detail (duplicates not omitted); **??** shows an exponential trend of API documentation publications produced within the last two decades.

Manual Filtering A follow-up manual filtering to select primary studies was performed on the 4,501 results using the following inclusion criteria (IC) and exclusion criteria (EC):

IC1 Studies must be relevant to API documentation: specifically, we exclude studies that deal with improving the technical API usability (e.g., improved usage patterns);

IC2 Studies must propose new knowledge or recommendations to document APIs;

IC3 Studies must be relevant to SE as defined in SWEBOK;

EC1 Studies where full-text is not accessible through standard institutional databases;

EC2 Studies that do not propose or extend how to improve the official, natural language documentation of an API;

[1] `http://apps.webofknowlegde.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.
[4] Raw results can be located at `http://bit.ly/2KxBLs4`

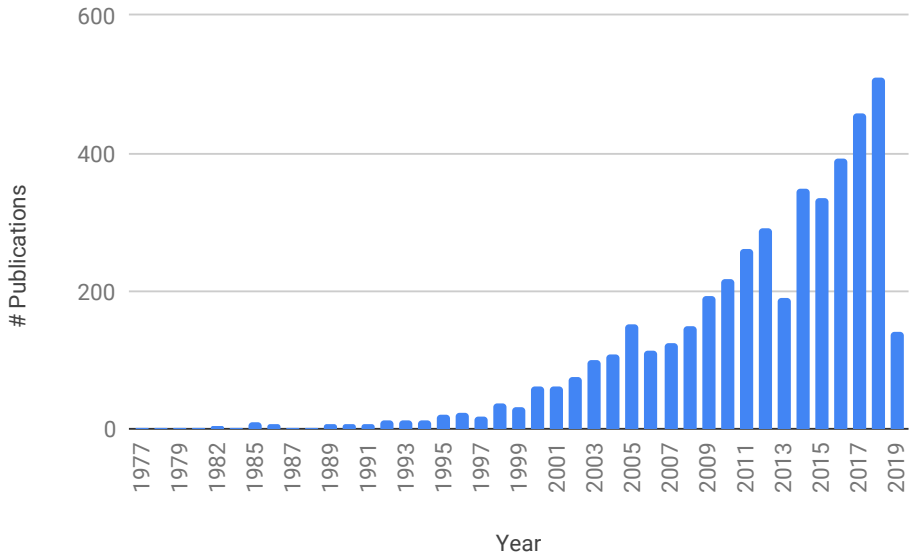


Figure 5.1: Histogram of the search results and years published. (As this search was conducted in May 2019, results taper in 2019.)

EC3 Studies proposing a third-party tool to enhance existing documentation or generate new documentation using data mining (i.e., not proposing strategies to improve official documentation);

EC4 Studies not written in English;

EC5 Studies not peer-reviewed.

After exporting metadata of search results to a spreadsheet, a three-phase curation process was conducted. The first author read the publication source (to omit non-SE papers quickly), author keywords and title of all 4,501 studies (514 that were duplicates), and abstract. As we considered multiple databases, some studies were repeated. However, the DOIs and titles were sorted and reviewed, retaining only one copy of the paper from a single database. Moreover, as there was no limit to the year range in our query, some studies were republished in various venues. These, too, were handled with title similarity matching, wherein only the first paper was considered. Where the inclusion or exclusion criteria could not be determined from the abstract alone, the paper was automatically shortlisted. Any doubt in a study automatically included it into the second phase. This resulted in 133 studies being shortlisted to the second phase. We rejected 427 studies that were unrelated to SE, 3,235 were not directly related to documenting APIs (e.g., to enhance coding techniques that improve the overall developer usability of the API), 182 proposed new tools to enhance API documentation or used machine learning to mine developer's discussion of APIs, and 10 were not in English.

The shortlisted studies were then re-evaluated by re-reading the abstract, the introduction and conclusion. Performing this second phase removed a further 64

studies that were on API usability or non API-related documentation (i.e., code commenting); we further refined our exclusion criteria to better match the research outcomes of this goal (chiefly including the word 'natural language' documentation in EC2) which removed studies focused to improve technical documentation of APIs such as data types and communication schemas. Additionally, 26 studies were removed as they were related to introducing new tools (EC3), 3 were focused on tools to mine API documentation, 7 studies where no recommendations were provided, 2 further duplicate studies, and a further 10 studies where the full text was not available, not peer reviewed or in English. Books are commonly not peer-reviewed (EC5), however no books were shortlisted within these results. This resulted in 21 primary studies for further analysis. The mapping of primary study identifiers to references S1–21 can be found in **??**.

Intra-rater reliability of our 133 shortlisted papers was tested using the test-retest approach [**?**] by re-evaluating a random sample of 10% (13 total) of the studies shortlisted above a week after initial studies were shortlisted. Using the Cohen's kappa coefficient as a metric for reliability, $\kappa = 0.7547$, indicating substantial agreement [**?**].

Data Extraction Of the 21 primary studies, we conducted abstract key-wording adhering to **?** 's guidelines [**?**] to develop a classification scheme. An initial set of keywords were applied for each paper in terms of their methodologies and research approaches (RQ2), based on an existing classification schema by **?**]: evaluation, validation, personal experience and philosophical papers.

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

?? describes our data extraction form, which was used to collect relevant data from each paper. **??** maps each study to one (or more, if applicable) of methodologies plotted against **?** 's research approaches.

5.3.2 Development of the Taxonomy

? concludes that a majority of SE taxonomies are developed in an ad-hoc way [**?**], and proposes a systematic approach to develop taxonomies in SE that extends previous efforts by including lessons learned from more mature fields. In this subsection, we outline the 4 phases and 13 steps taken to develop our taxonomy based on **?** 's technique.

Table 5.2: Data extraction form

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

Planning phase The planning phase of the technique involves the following six steps:

(1) *defining the SE KA*: The software engineering KA, as defined by the SWEBOK, is software construction;

(2) *defining the objective*: The main objective of the proposed taxonomy is to define a set of categories that enables to classify different facets of natural-language API *documentation* knowledge (not API *usability* knowledge) as reported in existing literature;

(3) *defining the subject matter*: The subject matter of our proposed taxonomy is documentation artefacts of APIs;

(4) *defining the classification structure*: The classification structure of our proposed taxonomy is *hierarchical*;

(5) *defining the classification procedure*: The procedure used to classify the documentation artefacts is qualitative;

(6) *defining the data sources*: The basis of the taxonomy is derived from field study techniques (see **??**).

Identification and extraction phase The second phase of the taxonomy development involves **(7)** *extracting all terms and concepts* from relevant literature, as selected from our 21 primary studies. These terms are then consolidated by **(8)** *performing terminology control*, as some terms may refer to different concepts and vice-versa.

Design phase The design phase identified the core dimensions and categories within the extracted data items. The first step is to **(9)** *identify and define taxonomy dimensions*; for this study we utilised a bottom-up approach to identify the dimensions, i.e., extracting the categories first and then nominating which dimensions

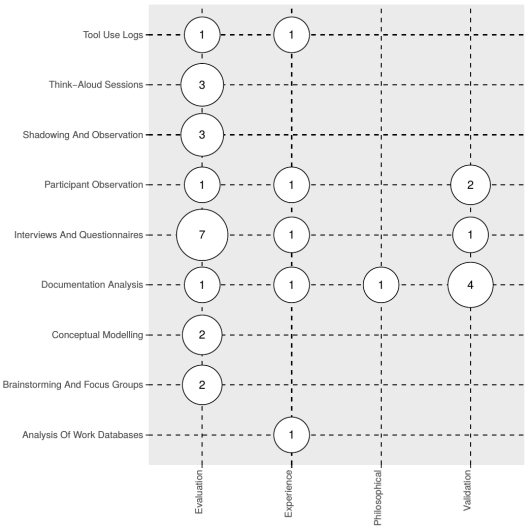


Figure 5.2: Systematic map: field study technique vs research type

these categories fit into using an iterative approach. As a bottom-up approach was utilised, step (9) also encompassed the second stage of the design phase, which is to **(10)** *identify and describe the categories* of each dimension. Thirdly, we **(11)** *identify and describe relationships* between dimensions and categories, which can be skipped if the relationships are too close together, as is the case of our grouping technique which allows for new dimensions and categories to be added. The last step in this phase is to **(12)** *define guidelines for using and updating the taxonomy*, however as this taxonomy still an emerging result, guidelines to update and use the taxonomy are anticipated future work.

Validation phase ⟨ TODO: *Revise this paragraph* ⟩ In the final phase of taxonomy development, taxonomy designers must **(13)** *validate the taxonomy* to assess its usefulness. **?**] describe three approaches to validate taxonomies: (i) orthogonal demonstration, in which the taxonomy's orthogonality is demonstrated against the dimensions and categories, (ii) benchmarking the taxonomy against similar classification schemes, or (iii) utility demonstration by applying the taxonomy heuristically against subject-matter examples. In our study, we adopt utility demonstration by use of a rigorous experiment and heuristic application against real-world case-studies (i.e., within the domain of IWSs). This is is discussed in greater detail within **??**.

5.4 Taxonomy Validation

⟨ TODO: *Describe validation* ⟩ To validate our taxonomy, we conducted a two-fold experiment. Firstly, we

5.4.1 Survey Study

5.4.2 Empirical Application against Computer Vision Services

5.5 Taxonomy

Our taxonomy consists of five dimensions (labelled A–E) that respectively cover:

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

Further descriptions of the categories encompassing each dimension are given within **??**, coded as $[Xi]$, where i is the category identifier within a dimension, X, where $X \in \{A, B, C, D, E\}$.

We expand these five dimensions into 34 categories (sub-dimensions) and **??** provides a weighting of these categories in the rightmost column as calculated as a percentage of the number of primary studies per category divided by the total of primary studies. The top five weighted categories (bolded in **??**) highlight what most studies recommend documenting in API documentation, with the top three falling under the Usage Description dimension.

The majority (71%) of studies advocate for **code snippets** as a necessary piece in the API documentation puzzle [A5]. While code snippets generally only reflect small portions of API functionality (limited to 15–30 LoC), this is complimented by **step-by-step tutorials** (57% of studies) that tie in multiple (disparate) components of API functionality, generally with some form of screenshots, demonstrating the development of a non-trivial application using the API step-by-step [A6]. The third highest category weighted was also under the Usage Description dimension, being **low-level reference documentation** at 52% [A2]. These three categories were the only categories to be weighted as majority categories (i.e., their weighting was above 50%).

The fourth and fifth highest weights are **an entry-level purpose/overview of the API** (48%) that gives a brief motivation as to why a developer should choose a particular API over another [B1] and **consistency in the look and feel** of the documentation throughout all of the API's official documentation (43%) [E6].

5.6 Threats to Validity

Threats to *internal validity* concern factors internal to our study that may affect results. Guidelines on producing systematic reviews [**?**] suggest that single

researchers conducting their reviews should discuss the review protocol, inclusion decisions, data extraction with a third party. In this paper, we have presented the early outcomes of our systematic review, which has utilised the test-retest methodology as a measure of reliability. **?**] states that a defining characteristic of any SMS is to test the reliability of the review and extraction processes. We plan to mitigate this threat by conducting *inter*-relater reliability with the continuation of this work, using independent analysis and conflict resolution as per guidelines suggested by **?**]. Similarly, the development of our taxonomy would benefit from an inter-rater reliability categorisation of a sample of papers to both ensure that our weightings of categories are reliable and that the categories and dimensions fit the objectives of the taxonomy. Furthermore, a future user study (see **??**) will be needed to assess whether the extracted information from API documentation actually impacts on developer productivity, and the usefulness of such a taxonomy should be evaluated.

Threats to *external validity* represent the generalisation of the observations we have found in this study. While we have used a broad range of literature that encompasses API documentation guidelines, we acknowledge that not all papers contributing to API documentation may have been captured in the taxonomy. All efforts were made to include as many papers as possible given our filtering technique, though it is likely that some papers filtered out (e.g., papers not in English) may alter our conclusions, introducing conflicting recommendations. However, given the consistency of these trends within the studies that were sourced, we consider this a low likelihood.

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

5.7 Conclusions & Future Work

API documentation is an aspect of quality of software, as it facilitates the developer's productivity and assists with evolution. Improving the quality of the documentation of third party APIs improves the quality of software using them.

To date, we did not find a systematic literature review that offers a consolidated taxonomy of key recommendations. Moreover, there has been little work on mapping the research produced in this space against the techniques used to arrive at the recommendations. Starting with 4,501 papers potentially relating to API documentation, we identified 21 key relevant studies, and synthesise a taxonomy of the various documentation aspects that should improve API documentation quality. Furthermore, we also capture the most commonly used analysis techniques used in the

academic literature. **??** highlights that a majority of these studies employ interviews and questionnaires, and only some undertake structured documentation analysis.

In future revisions of this work, we intend use our results as the input to a restricted systematic literature review in API documentation artefacts. In doing so, we will consider conducting the following:

- improving reliability metrics of our study (see **??**) with an inter-rater reliability method;
- reviewing the techniques and evaluation of our selected studies to extract the effectiveness of the various approaches used in the conclusions

We believe the results of this preliminary empirical work may provide further insight for future follow-up user studies with developers. Whilst our aim is to eventually improve the quality of API documentation, the ultimate goal is improving the developer's experience when producing systems and, therefore, improving the efficacy and productivity at which software is produced within industry. We hope that API designers will utilise the taxonomy produced in this paper as a weighted checklist for what should be considered in their own APIs.

Table 5.3: An overview of the 5 dimensions and categories (sub-dimensions) within our proposed taxonomy.

Key	Description	Primary Sources	Total (%)
A1	Quick-start guides to rapidly get started using the API in a specific programming language.	S4, S9, S10	3/21 (14%)
A2	Low-level reference manual documenting all API components to review fine-grade detail.	S1, S3, S4, S8, S9, S10, S11, S12, S15, S16, S17	11/21 (52%)
A3	Explanations of the API's high-level architecture to better understand intent and context.	S1, S2, S4, S11, S14, S16, S19, S20	8/21 (38%)
A4	Source code implementation and code comments (where applicable) to understand the API author's mindset.	S1, S4, S7, S12, S13, S17, S20	7/21 (33%)
A5	Code snippets (with comments) of no more than 30 LoC to understand a basic component functionality within the API.	S1, S2, S4, S5, S6, S7, S9, S10, S11, S14, S15, S16, S18, S20, S21	15/21 (71%)
A6	Step-by-step tutorials, with screenshots to understand how to build a non-trivial piece of functionality with multiple components of the API.	S1, S2, S4, S5, S7, S9, S10, S15, S16, S18, S20, S21	12/21 (57%)
A7	Downloadable source code of production-ready applications that use the API to understand implementation in a large-scale solution.	S1, S2, S5, S9, S15	5/21 (24%)

Continued on next page...

An overview of the 5 dimensions and categories (sub-dimensions) within our proposed taxonomy *(Continued)*.

Key	Description	Primary Sources	Total (%)
A8	Best-practices of implementation to assist with debugging and efficient use of the API.	S1, S2, S4, S5, S7, S8, S9, S14	8/21 (38%)
A9	An exhaustive list of all major components that exist within the API.	S4, S16, S19	3/21 (14%)
A10	Minimum system requirements and dependencies to use the API.	S4, S7, S13, S17, S19	5/21 (24%)
A11	Instructions to install or begin using the API and details on its release cycle and updating it.	S4, S7, S8, S9, S11, S13, S16, S19	8/21 (38%)
A12	Error definitions that describe how to address a specific problem.	S1, S2, S4, S5, S9, S11, S13	7/21 (33%)
B1	A brief description of the purpose or overview of the API as a low barrier to entry.	S1, S2, S4, S5, S6, S8, S10, S11, S15, S16	10/21 (48%)
B2	Descriptions of the types of applications the API can develop.	S2, S4, S9, S11, S15, S18	6/21 (29%)
B3	Descriptions of the types of users who should use the API.	S4, S9	2/21 (10%)
B4	Descriptions of the types of users who will use the product the API creates.	S4	1/21 (5%)
B5	Success stories about the API used in production.	S4	1/21 (5%)
B6	Documentation to compare similar APIs within the context to this API.	S2, S6, S13, S18	4/21 (19%)
B7	Limitations on what the API can and cannot provide.	S4, S5, S8, S9, S14, S16	6/21 (29%)
C1	Descriptions of the relationship between API components and domain concepts.	S3, S10	2/21 (10%)
C2	Definitions of domain-terminology and concepts, with synonyms if applicable.	S2, S3, S4, S6, S7, S10, S14, S16	8/21 (38%)
C3	Generalised documentation for non-technical audiences regarding the API and its domain.	S4, S8, S16	3/21 (14%)
D1	A list of FAQs.	S4, S7	2/21 (10%)
D2	Troubleshooting suggestions.	S4, S8	2/21 (10%)
D3	Diagrammatically representing API components using visual architectural representations.	S6, S13, S20	3/21 (14%)
D4	Contact information for technical support.	S4, S8, S19	3/21 (14%)
D5	A printed/printable resource for assistance.	S4, S6, S7, S9, S16	5/21 (24%)
D6	Licensing information.	S7	1/21 (5%)

Continued on next page...

An overview of the 5 dimensions and categories (sub-dimensions) within our proposed taxonomy *(Continued)*.

Key	Description	Primary Sources	Total (%)
E1	Searchable knowledge base.	S3, S4, S6, S10, S14, S17, S18	7/21 (33%)
E2	Context-specific discussion forum.	S4, S10, S11	3/21 (14%)
E3	Quick-links to other relevant documentation frequently viewed by developers.	S6, S16, S20	3/21 (14%)
E4	Structured navigational style (e.g., bread-crumbs).	S6, S10, S20	3/21 (14%)
E5	Visualised map of navigational paths to certain API components in the website.	S6, S14, S20	3/21 (14%)
E6	Consistent look and feel of documentation.	S1, S2, S3, S5, S6, S8, S10, S15, S20	9/21 (43%)

CHAPTER 6

Interpreting Pain-Points in Computer Vision Services[†]

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

6.1 Introduction

The availability of recent advances in artificial intelligence (AI) over simple RESTful end-points offers application developers new opportunities. These new intelligent services (AI components) abstract complex machine learning (ML) and AI techniques behind simpler API calls. In particular, they hide (either explicitly or

[†]This chapter is originally based on . Terminology has been updated to fit this thesis.

implicitly) any data-driven and non-deterministic properties inherent to the process of their construction. The promise is that software engineers can incorporate complex machine learnt capabilities, such as computer vision, by simply calling an API end-point.

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

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

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

This study involves an investigation of 1,825 Stack Overflow (SO) posts regarding

one of the most mature types of intelligent services—computer vision services—dating from November 2012 to June 2019. We adapt existing methodologies of prior SO analyses [**?** **?**] to extract posts related to computer vision services. We then apply two existing SO question classification schemes presented at ICPC and ICSE in 2018 and 2019 [**?** **?**]. These previous studies focused on mobile apps and web applications. Although not a direct motivation, our work also serves as a validation of the applicability of these two issue classification taxonomies [**?** **?**] in the context of intelligent services (hence potential for generalisation). Additionally our work is the first—to our knowledge—to *test* the applicability of these taxonomies in a new study.

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

The key findings of our study are:

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

The paper also offers a data-set as an additional contribution to the research community and to permit replication [**?**]. The paper structure is as follows: **??** provides motivational examples to highlight the core focus of our study; **??** provides a background on prior studies that have mined SO to gather insight into the SE community; **??** describes our study design in detail; **??** presents the findings from the SO extraction; **??** offers an interpretation of the results in addition to potential implications that arise from our work; **??** outlines the limitations of our study; concluding remarks are given in **??**.

6.2 Motivation

"Intelligent" services are often available as a cloud end-point and provide developers a friendly approach to access recent AI/ML advances without being experts in the underlying processes. **??** highlights how these services abstract away much of the technical know-how needed to create and operationalise these intelligent services [**?**]. In particular, they hide information about the training algorithm and data-sets used in training, the evaluation procedures, the optimisations undertaken, and—

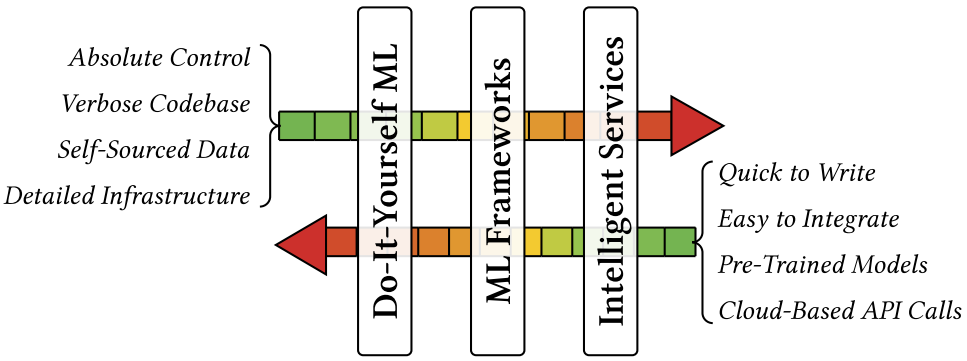


Figure 6.1: Some traits of Intelligent Services vs. 'Do-It-Yourself' ML. Green-to-red arrows indicate the presence of these traits. *Adapted from* **?** *].*

surprisingly—they often do not offer a properly versioned end-point [**?** **?**]. That is, the cloud vendors may change the behaviour of the services without sufficient 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 **??**). 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.*" [**?**]; "*I'm trying out the [intelligent service], and there's a score field that returns that I'm not sure how to interpret [it].*" [**?**]
- **frustrated about non-deterministic results:** "*Often the API has troubles in recognizing single digits... At other times Vision confuses digits with letters.*" [**?**]; "*Is there a way to help the program recognize numbers better, for example limit the results to a specific format, or to numbers only?*" [**?**]
- **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.*" [**?**]
- **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.*" [**?**]

The objective of our study is to better understand the nature of the questions that developers raise when using intelligent services, 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 [**?** **?** **?**]. We build on top of this work by adapting the study methodology and apply the taxonomies offered to

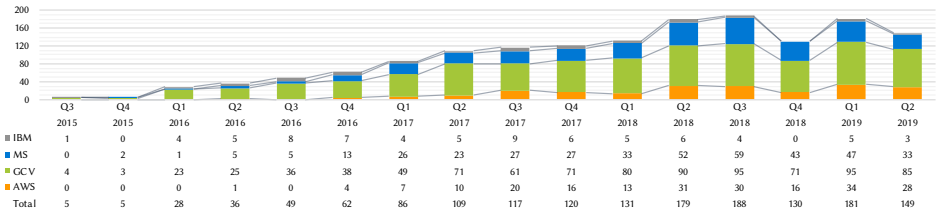


Figure 6.2: Trend of posts, where IBM = IBM Watson Visual Recognition, MS = Azure Computer Vision, AWS = AWS Rekognition and GCV = Google Cloud Vision. Three MS posts from Q4 2012, Q3 2013 and Q4 2013 have been removed for graph clarity.

identify the nature of the issues and this results in the following research questions in this paper:

RQ1. RQ1. How do developers mis-comprehend intelligent services as presented within Stack Overflow pain-points? While the AI community is well aware in the the nuances that empower intelligent services, such services are being released for application developers who may not be aware of their limitations or how they work. This is especially the case when machine intelligence is accessed via web-based APIs where such details are not fully exposed.

RQ2. Are the distribution of issues similar to prior studies? We compare how the distributions of previous studies' of posts about conventional, deterministic API services differ from those of intelligent services. By assessing the distribution of intelligent services' issues against similar studies that focus on mobile and web development, we identify whether a new taxonomy is needed specific to AI-based services, and if gaps specific to AI knowledge exist that need to be captured in these taxonomies.

6.3 Background

The primary goal of analysing issues is to better understand the root causes. Hence, a good issue classification taxonomy should ideally capture the underlying causal aspects (instead of pure functional groupings) [**?**]. Although this idea (of cause related classification) is not new (Chillarege advocated for it in this TSE paper in 1992), this is not a universally followed approach when studying online discussions and some recent works have largely classified issues into the *"what is"* and not *"how to fix it"* [**?** **?** **?**]. They typically (manually) classify discussion into either *functional areas* (e.g., Website Design/CSS, Mobile App Development, .NET Framework, Java [**?**]) or *descriptive areas* (e.g., Coding Style/Practice, Problem/Solution, Design, QA [**?** **?**]). As a result, many of these studies do not give us a prioritised means of targeted attack on how to *resolve* these issues with, for example, improved documentation. Interestingly, recent taxonomies that studied SO data (**?**] and **?**]) were causal in nature and developed to understand discussions related to mobile and web applications. However, issues that arise when developers use intelligent services

have not been studied, nor do we know if existing issue classification taxonomies are sufficient in this domain.

Researchers studying APIs have also attempted to understand developer's opinions towards APIs [**?**], categorise the questions they ask about these APIs [**? ? ? ?**], and understand API related documentation and usage issues [**? ? ? ? ? ?**]. These studies often employ automation to assist in the data analysis stages of their research. Latent Dirichlet Allocation [**? ? ? ?**] is applied for topic modelling and other ML techniques such as Random Forests [**?**], Conditional Random Fields [**?**] or Support Vector Machines [**? ?**] are also used.

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

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

6.4 Method

6.4.1 Data Extraction

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

However, during the filtering stage (see **??**), we decided to focus on a subset of these services, computer vision, as these are one of the more mature and stable ML/AI-based services with widespread and increasing adoption in the developer community (see **??**). We acknowledge other services beyond the four analysed provide similar capabilities [**? ? ? ? ? ?**] and only English-speaking services have been selected, excluding popular services from Asia (e.g., [**? ? ? ? ?**])—see **??**. For comprehensiveness, we explain below our initial attempts to extract *all* intelligent services.

Defining a list of intelligent services As there exists no global 'list' of intelligent services to search on, we needed to derive a *corpus of initial terms* to allow us to know *what* to search for on the Stack Exchange Data Explorer[1] (SEDE). We began by looking at different brand names of cloud services and their permutations (e.g., Google Cloud Services and GCS) as well as various ML-related products (e.g., Google Cloud ML). To do this, we performed extensive Google searches[2] in addition to manually reviewing six 'overview' pages of the relevant cloud platforms. We identified 91 initial intelligent services to incorporate into our search terms[3].

Manual search for relevant, related terms We then ran a manual search[2] on each term to determine if these terms were relevant. We did this by querying each term within SO's search feature, reviewing the titles and body post previews of the first three pages of results (we did not review the answers, only the questions). We also noted down the user-defined *Tags* of each post (up to five per question); by clicking into each tag, we could review similar tags (e.g., 'project-oxford' for 'azure-cognitive-services') and check if the tag had synonyms (e.g., 'aws-lex' and 'amazon-lex'). We then compiled a *corpus of tags* consisting of 31 terms.

Developing a search query We recognise that searching SEDE via *Tags* exclusively can be ineffective (see [**? ?**]). To mitigate this, we produced a *corpus of title and body terms*. Such terms are those that exist within the title and body of the posts to reflect the ways in which individual developers commonly use to refer to different intelligent services. To derive at such a list, we performed a search[2,3] of the 31 tags above in SEDE, filtering out posts that were not answers (i.e., questions only) as we wanted to see how developers *phrase* their questions. For each search, we extracted a random sample of 100 questions (400 total for each service) and reviewed each question. We noted many patterns in the permutations of how developers refer to these services, such as: common misspellings ('bind' vs. 'bing'); brand misunderstanding ('Microsoft computer vision' vs. 'Azure computer vision'); hyphenation ('Auto-ML' vs. 'Auto ML'); UK and US English ('Watson Analyser' vs. 'Watson Analyzer'); and, the use of apostrophes, plurals, and abbreviations ('Microsoft**'s** Computer Vision API', 'Microsoft Computer Vision Service**s**, 'GCV' vs. 'Google Cloud Vision'). We arrived at a final list of 229 terms compromising all of the intelligent services provided by Google, Amazon, Microsoft and IBM as of January 2019[3].

Executing our search query Our next step was to perform a case-insensitive search of all 229 terms within the body or title of posts. We used Google BigQuery's public data-set of SO posts[4] to overcome SEDE's 50,000 row limit and to conduct a case-insensitive search. This search was conducted on 10 May 2019, where we

[1] `http://data.stackexchange.com/stackoverflow`

[2] This search was conducted on 17 January 2019

[3] For reproducibility, this is available at `http://bit.ly/2ZcwNJ0`.

[4] http://bit.ly/2LrN7OA

extracted 21,226 results. We then performed several filtering steps to cleanse our extracted data, as explained below.

6.4.2 Data Filtering

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

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

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

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

Classification Our 1,425 posts were then split into 4 additional random samples (in addition to the random sample of 50 above). 475 posts were classified by the first author and three other research assistants, software engineers with at least 2 years industry experience, assisted to classify the remaining 900. This left a total of 1,375 classifications made by four people plus an additional 450 classifications made from

[5]Watson Cognitive Services

[6]We make this available for future use at: `http://bit.ly/2NqBB70`

reliability analysis, in which the remaining 50 posts were classified nine times (as detailed in **??**). Thus, a total of 1,825 classifications were made from the original 1,425 posts extracted.

Whilst we could have chosen to employ topic modelling, these are too descriptive in nature (as discussed in **??**). Moreover, we wanted to see if prior taxonomies can be applied to intelligent services (as opposed to creating a new one) and compare if their distributions are similar. Therefore, we applied the two existing taxonomies described in **??** to each post; (i) a documentation-specific taxonomy that addresses issues directly resulting from documentation, and (ii) a generalised taxonomy that covers a broad range of SO issues in a well-defined SE area (specifically mobile app development). Aghajani et al.'s documentation-specific taxonomy (Taxonomy A) is multi-layered consisting of four dimensions and 16 sub-categories [**?**]. Similarly, Beyer's SO generalised post classification taxonomy (Taxonomy B) consists of seven dimensions [**?**]. We code each dimension with a number, X, and each sub-category with a letter y: (Xy). We describe both taxonomies in detail within **??**. Where a post was included in our results but not applicable to intelligent services (see **??**) or not applicable to a taxonomy dimension/category, then the post was flagged for removal in further analysis. **??** presents *our understanding* of the respective taxonomies; our intent is not to methodologically replicate **?** or **?** 's studies in the intelligent service domain, rather to acknowledge related work in the area of SO classification and reduce the need to synthesise a new taxonomy. We baseline all coding against *our interpretation only*. Our classifications are therefore independent of the previous authors' findings.

Table 6.1: Descriptions of dimensions (■) and sub-categories (↪) from both taxonomies used.

A	**Documentation-specific classification (?])**	
A-1	■ **Information Content (What)**	Issues related to what is written in the documentation
A-1a	↪ ***Correctness***	What exists in the documentation actually matches what is implemented in code
A-1b	↪ ***Completeness***	The documentation fully covers all aspects of the API's components
A-1c	↪ ***Up-to-dateness***	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	↪ ***Maintainability***	The upkeep effort to ensure the documentation remains up to date
A-2b	↪ ***Readability***	The extent to which the documentation is interpretable
A-2c	↪ ***Usability***	How useable the organisation, look and feel of the documentation is
A-2d	↪ ***Usefulness***	The usefulness of the documentation, avoiding misinformation.
A-3	■ **Process-Related**	Issues related to the documentation process
A-3a	↪ ***Internationalisation***	Translating the documentation into other languages
A-3b	↪ ***Contribution-Related***	Contribution issues encountered when people contribute to the documentation
A-3c	↪ ***Configuration-Related***	Configuration issues of the documentation tool
A-3d	↪ ***Implementation-Related***	Unwanted development issues caused by (poor) documentation
A-3e	↪ ***Traceability***	Tracing documentation changes (when, when, who and why)
A-4	■ **Tool-Related**	Issues related to documentation tools (e.g., Javadoc)
A-4a	↪ ***Tooling Bugs***	Bugs that exist within the documentation tooling
A-3b	↪ ***Tooling Discrepancy***	Support as expectations not being fulfilled by these documentation tools
A-3c	↪ ***Tooling Help Required***	Help required due to improper usage of the tools
A-3d	↪ ***Tooling Migration***	Issues migrating the tool to a new version or another tool
B	**Generalised classification (?])**	
B-1	■ **API usage**	Issue on how to implement something using a specific component provided by the API
B-2	■ **Discrepancy**	The questioner's *expected behaviour* of the API does not reflect the API's *actual behaviour*
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

6.4.3 Data Analysis

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

Firstly, intra-rater agreement by the first author was conducted twice on 28 June 2019 and 9 August 2019. Secondly, inter-rater agreement was conducted with the remaining four co-authors in addition to three research assistants within our research group in mid-August 2019. Thus, the 50 posts were classified an additional nine times, resulting in 450 classifications for reliability analysis. We include these classifications in our overall analysis.

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

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

6.5 Findings

We present our findings from classifying a total of 1,825 SO posts aimed at answering RQs 1 and 2. 450 posts were classified using Taxonomies A and B for reliability analysis as described in **??** and the remaining 1,375 posts were classified as per **??**. A summary of our classification using Taxonomies A and B is shown in **??**.

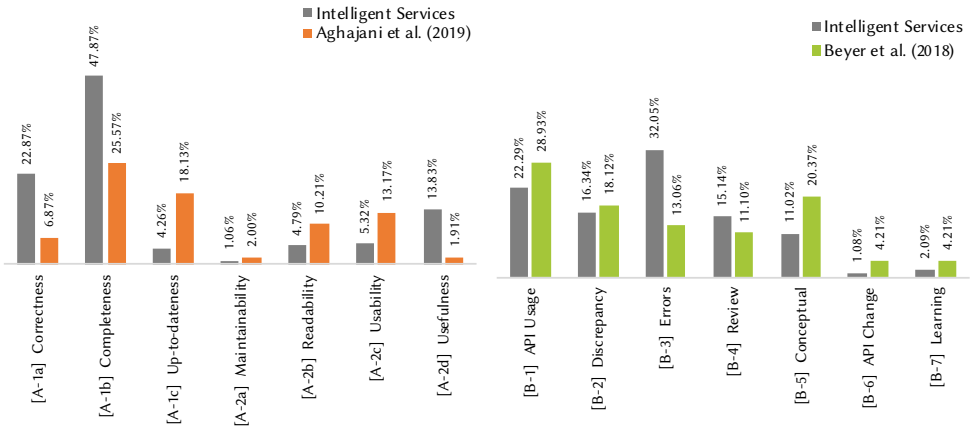


Figure 6.3: *Left:* Documentation-specific classification taxonomy results highlights a mostly similar distribution to that of **?** 's findings [**?**]. *Right:* Generalised classification taxonomy results highlight differences from more mature fields (i.e., Android APIs in **?**]) to less mature fields (i.e., intelligent services).

6.5.1 Post classification and reliability analysis

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

For our reliability analysis, we classified a total of 450 posts of which 70 posts were flagged as irrelevant. **?**] provide guidelines to interpret kappa reliability statistics, where $0.00 \leq \kappa \leq 0.20$ indicates *slight* agreement and $0.21 \leq \kappa \leq 0.40$ indicates *fair* agreement. Despite all raters meeting to agree on a shared interpretation of the taxonomies (see **??**) our inter-rater measures aligned *slightly* (0.148) for Taxonomy A and *fairly* (0.295) for Taxonomy B. We report further in **??**.

6.5.2 Developer Frustrations

We found **?** 's high-level abstraction taxonomy (Taxonomy B) was able to classify 86.52% of posts. 10.30% posts were assigned exclusively under **?** '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.

Results from ? 's taxonomy Results for **?** 's low-level documentation taxonomy (Taxonomy A), indicates that most discussion on SO does not directly relate to documentation about an intelligent service. 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 intelligent services 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 intelligent services.

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?*" [**?**]; "*There seems to be very little to no documentation for AWS iOS text recognition inside an image*" [**?**].

22.87% of posts question the **accuracy (A-1a)** of certain parts of the cloud documentation, especially in relation to incorrect quotas and limitations: "*Are the Cloud Vision API limits in documentation correct?*" [**?**], "*According to the Google Vision documentation, the maximum number of image files per request is 16. Elsewhere, however, I'm finding that the maximum number of requests per minute is as high as 1800.*" [**?**].

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

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

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

It was commonplace to see questions that demonstrate a lack of depth in understanding and appreciating how these services work, instead posting simple debugging questions. For instance, in the 11.02% of **conceptual-related questions (B-5)** that we categorised, we noticed causal links to a misunderstanding (or lack of awareness) of the vocabulary used within computer vision. For example: "*The problem is that I*

need to know not only what is on the image but also the position of that object. Some of those APIs have such feature but only for face detection." [**?**]; "*I want to know if the new image has a face similar to the original image.... [the service] can identify faces, but can I use it to get similar faces to the identified face in other images?*" [**?**]. It is evident that some application developers are not aware of conceptual differences in computer vision such as object/face *detection* versus *localisation* versus *recognition*.

In the 16.34% of **discrepancy-related questions (B-2)**, we see further unawareness from developers in how the underlying systems work. In OCR-related questions, developers do not understand the pre-processing steps required before an OCR is performed. In instances where text is separated into multiple columns, for example, text is read top-down rather than left-to-right and segmentation would be required to achieve the expected results. For example, "*it appears that the API is using some kind of logic that makes it scan top to bottom on the left side and moving to right side and doing a top to bottom scan.*" [**?**]; "*this method returns scanned text in wrong sequence... please tell me how to get text in proper sequence.*" [**?**].

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

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

6.5.3 Statistical Distribution Analysis

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

6.6 Discussion

6.6.1 Answers to Research Questions

RQ1. How do developers mis-comprehend intelligent services as presented within Stack Overflow pain-points? Upon meeting to discuss the discrepancies between our categorisation of intelligent service usage SO posts, we found that our interpretations of the *posts themselves* were largely subjective. For example, many posts presented multi-faceted dimensions for Taxonomy B; **?**] argue that a post can have more than one question category and therefore multi-label classification is appropriate at times. We highlight this further in the threats to validity (**??**).

We have to define the context of intelligent services to address RQ1. We use the concept of a "technical domain" [**?**] to define this context. A technical domain captures the domain-specific concerns that influence the non-functional requirements of a system [**?**]. In the context of intelligent services, the technical domain includes exploration, data engineering, distributed infrastructure, training data, and model characteristics as first class citizens [**?**]. We would then expect to see posts on SO related to these core concerns.

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

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

RQ2. Are the distribution of issues similar to prior studies? Visual inspection of **??** shows that the distributions for the documentation-specific classification and the generalised classification are different (compared to prior studies). As a sanity check we conducted a χ^2 test and calculated the effect size ϕ_c. We were able to reject the null hypothesis for both classification schemes, that the distribution of issues were the same as the previous studies (see **??**). We now discuss the most prominent differences between our study and the previous studies.

In the context of intelligent service SO posts, Taxonomy B suggests that Errors (B-3) are discussed most amongst developers. These results are in contrast to similar studies made in more *mature* API domains, such as Mobile Development [**? ? ? ?**

?] and Web Development [**?**]. Here, API Usage (B-1) is much more frequently discussed, followed by Conceptual (B-5), Discrepancy (B-2) and Errors (B-3). We argue in the following section that an improved developer understanding can be achieved by educating them about the intelligent service lifecycle and the 'whole' system that wraps such services.

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

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

6.6.2 The Developer's Learning Approach

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

Bloom's Taxonomy The cognitive domain under Bloom's taxonomy [**?**] consists of six objectives. Within the context of intelligent services, developers are likely to ask questions due to causal links that exist in the following layers of Bloom's taxonomy: (i) *knowledge*, where the developer does not remember or know of the basic concepts of computer vision and AI (in essence, they may think that AI is as smart as a human); (ii) *comprehension*, where the developer does not understand how to interpret basic concepts, or they are mis-understanding how they are used

in context; (iii) *application*, where the developer is struggling to apply existing concepts within the context of their own situation; (iv) *analysis*, where the developer is unable to analyse the results from intelligent services (i.e., understand response objects); (v) *evaluation*, where the developer is unable to evaluate issues and make use of best-practices when using intelligent services; and (vi) *synthesise*, where the developer is posing creative questions to ask if new concepts are possible with computer vision services.

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

Table 6.2: Example Alignments of Stack Overflow 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.*" [**?**]	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).*" [**?**]	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?*" [**?**]	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?*" [**?**]	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?*" [**?**]	Analysis	Relational

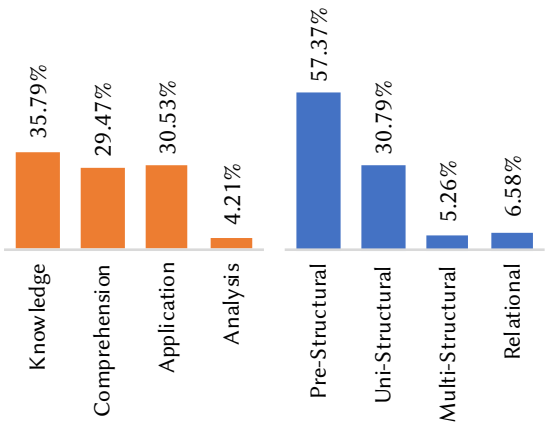


Figure 6.4: Alignment of Bloom (Orange) and SOLO (Blue) taxonomies against Taxonomy A and B dimensions against all 213 classifications made in the random sample of 50 posts.

Aligning SO taxonomies to Bloom's and SOLO taxonomies To understand our findings with the lenses of pedagogical aids, we aligned Taxonomies A and B to Bloom's and the SOLO taxonomies for a random sample of 50 issues described in **??**. To do this, we reviewed all 50 of these SO posted 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 intelligent services) 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 **??**. We acknowledge that this is only an approximation of the current state of the developer's understanding of intelligent services. This early model will require further studies to perform a more thorough analysis, but we offer this interpretation for early discussion.

As shown in **??**, 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?*" [**?**]; "*I am a 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.*" [**?**]

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.

6.6.3 Implications

Researchers *(i) Investigate the evolution of post classification:* Analysing how the distribution of the reported issues changes over time would be an important study. This study could answer questions such as '*Does the evolution of intelligent services follow the same pattern as previous software engineering trends such as mobile app or web development?*' As with any new emerging field, it is key to analyse how developers perceive such issues over time. For instance, early issues with web or mobile app development matured as their respective domain matured, and we would expect similar results to occur in the intelligent services space. Future researchers could plan for a longitudinal study, such as a long-term survey with developers to gather their insights in this evolving domain, reviewing case studies of projects that use intelligent web services from now into the future, or re-mining SO at a later date and comparing the results to this study. This will help assess evolving trends and characteristics, and determine how and if the nature of the developer's experience with intelligent services (and AI in general) changes with time. *(ii) Investigate the impact of technical challenges on API usage:* As discussed above, intelligent services have characteristics that may influence API usage patterns and should be investigated as a further avenue of research. Further mining of open source software repositories that make use of intelligent services could be assessed, thereby investigating if API patterns evolve with the rise of AI-based applications.

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

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

Intelligent Service Providers *(i) Clarify use cases for intelligent services:* Inspecting SO posts revealed that there is a level of confusion around the capabilities of different intelligent services. This needs to be clarified in associated API documentation. The complication with this comes with targeting the documentation such that software developers (who are untrained in the nuances of AI and ML as per **??**) can to digest it and apply it in-context to application development. *(ii) Technical domain matters:* More needs to be provided than a simple endpoint description as conventional APIs offer by describing the whole framework by which the endpoint sits, giving further context. This said, compared to traditional APIs, we find that developers complain less about the documentation and more about shallower issues. All expected pre-processing and post-processing needs to be clearly explained. A possible mitigation to this could be an interactive tutorial that helps developers fully understand the technical domain using a hands-on approach. For example, websites offer interactive Git tutorials[7] to help developers understand and explore the technical domain matters under version control in their own pace. *(iii) Clarify limitations:* API developers need to add clear limitations of the existing APIs. Limitations include list of objects that can be returned from an endpoint. We found that the cognitive anchors of how existing, conventional API documentation is written has become 'ported' to the computer vision realm, however a lot more overview documentation than what is given at present (i.e., better descriptions of errors, improved context of how these systems work in etc.) needs to be given. Such documentation could be provided using interactive tutorials.

[7]For example, `https://learngitbranching.js.org`.

6.7 Threats to Validity

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

Internal validity: As detailed in **??**, Taxonomies A and B present slight and fair agreement, respectively, when inter-rater reliability was applied. The nature of our disagreements largely fell due to the subjectivity in applying either taxonomies to posts. Despite all coders agreeing to the shared interpretation of both taxonomies, both taxonomies are subjective in their application, which was not reported by either **?** or **?** . In many cases, multi-label classification seemed appropriate, however both taxonomies use single-label mapping which we find results in too much subjectivity. This subjectivity, therefore, ultimately adversely affects IRR analysis. Thus, a future mitigation strategy for similar work should explore multi-label classification to avoid this issue; **?** , for example, plan for multi-label classification as future work. However, these studies would need to consider the statistical challenges in calculating multi-rater, multi-label IRR for thorough reliability analysis in addressing subjectivity. The selection of SO posts used for our labelling, chiefly in the subjectivity of our classifications, is of concern. We mitigate this by an extensive review process assessing the reliability of our results as per **??**. The classification of our posts into the SOLO and Bloom's taxonomies was performed by the primary author only, and therefore no inter-rater reliability statistics were performed. However, we used these pedagogy related taxonomies as a lens to gain an additional perspective to interpret our results. Future studies should attempt a more rigorous analysis of SO posts using Bloom's and SOLO taxonomies. We only aligned posts to one category for each taxonomy and did not align these using multi-label classification. This brings more complexity to the analysis, and our attempts to repeat prior studies' methodologies (see **??**). Multi-label classification for intelligent services SO posts is an avenue for future research.

External validity: While every effort was made to select posts from SO relevant to computer vision services, there are some cases where we may have missed some posts. This is especially due to the case where some developers mis-reference certain intelligent services under different names (see **??**). Our SOLO and Bloom's taxonomy analysis has only been investigated through the lenses of intelligent services, and not in terms of conventional APIs (e.g., Andriod APIs). Therefore, we are not fully certain how these results found would compare to other types of APIs. Two *existing* SO classification taxonomies were used rather than developing our own. We wanted to see if previous SO taxonomies could be applied to intelligent services before developing a new, specific taxonomy, and these taxonomies were applied based on our interpretation (see **??**) and may not necessarily reflect the interpretation of the original authors. Moreover, automated techniques such as topic modelling were not

utilised as we found these produce descriptive classifications only (see **??**). Hence, manual analysis was performed by humans to ensure categories could be aligned back to causal factors. Only English-speaking intelligent services were selected; the applicability of our analysis to other, non-English speaking services may affect results. Use of computer vision in this study is an illustrative example to focus on one area of the intelligent services spectrum. While our narrow scope helps us obtain more concrete findings, we suggest that wider issues exist in other intelligent service domains may affect the genralisability of this study, and suggest future work be explored in this space.

6.8 Conclusions

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

CHAPTER 7

Ranking Computer Vision Service Issues using Emotion[†]

[†]This chapter is originally based on . Terminology has been updated to fit this thesis.

CHAPTER 8

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

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

8.1 Introduction

With the introduction of intelligent web services (IWSs) that make machine learning (ML) more accessible to developers [**? ?**], we have seen a large growth of intelligent applications dependent on such services [**? ?**]. For example, consider the advances made in computer vision, where objects are localised within an image and labelled with associated categories. Cloud-based computer vision services (CVSs)—e.g., [**? ? ? ? ? ? ? ?**]—are a subset of IWSs. They utilise ML techniques to achieve image recognition via a remote black-box approach, thereby reducing the overhead for application developers to understand how to implement intelligent systems from scratch. Furthermore, as the processing and training of the machine-learnt algorithms is offloaded to the cloud, developers simply send RESTful API requests to do the recognition. There are, however, inherit differences and drawbacks between traditional web services and IWSs, which we describe with the motivating

[†]This chapter is originally based on . Terminology has been updated to fit this thesis.

scenario below.

8.1.1 Motivating Scenario: Intelligent vs Traditional Web Services

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

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

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

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

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

(3) Intelligent APIs evolve over time. Tom may find that responses to processing an image may change over time; the labels he processes in testing may evolve and therefore differ to when in production. In traditional web services, evolution in responses is slower, generally well-communicated, and usually rare (Tom would always expect `"dog"` to be returned in the substring example).

This has many implications on software systems that depend on these APIs, such as confidence in the output and portability of the solution. Currently, if Tom switches from one API provider to another, or if he doesn't regularly test his app in production, he may begin to see a very different set of labels and confidence levels.

8.1.2 Research Motivation

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

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

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

Previous attempts at overcoming low reliability include triple-modular redundancy [**?**]. This method uses three modules and decides output using majority rule. However, in CVSs, it is difficult to apply majority rule: these APIs respond with a list of labels and corresponding scores. Moreover, disparate APIs ordinarily output different results. These differences make it hard to apply majority rule because the type of outputs are complex and disparate APIs output different results for the same input. Merging search results is another technique to improve reliability [**?**]. It normalises scores of different databases using a centralised sample database. Normalising scores makes it possible to merge search results into a single ranked list. However, search responses are disjoint, whereas they are not in the context of most CVSs.

In this paper, we introduce a novel method to merge responses of CVSs, using image recognition APIs endpoints as our motivating example. **??** describes naive merging methods and requirements. **??** gives insights into the structure of labels. **??** introduces our method of merging computer vision labels. **??** compares precision and recall for each method. **??** presents conclusions and future work.

8.2 Merging API Responses

Image recognition APIs have similar interfaces: they receive a single input (image) and respond with a list of labels and associated confidence scores. Similarly, other supervised-AI-based APIs do the same (e.g., detecting emotions from text and natural language processing [**?** **?**]). It is difficult to apply majority rule on such disparate, complex outputs. While the outputs by *multiple* AI-based API endpoints is different and complex, the general format of the output is the same: a list of labels and associated scores.

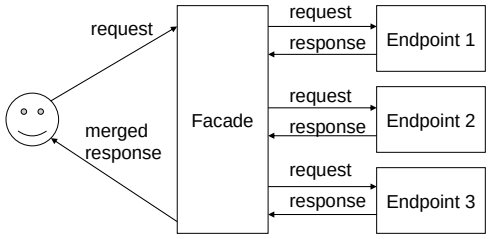


Figure 8.1: The user sends a request to the facade; this request is propagated to the relevant APIs. Responses are merged by the facade and returned back to the user.

8.2.1 API Facade Pattern

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

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

- **Easy to modify:** It requires only small modifications to applications, e.g., changing each concrete endpoint URL.
- **Easy to customise:** It merges results from disparate and concrete APIs according to the user's preference.
- **Improves reliability:** It enhances reliability of the overall returned result by merging results from different endpoints.

8.2.2 Merge Operations

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

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

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

```
[['apple', 0.7], ['cherry', 0.8]].
```

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

`[['apple', 0.9], ['banana', 0.8], ['cherry', 0.8]];`

min merges results to:

`[['apple', 0.7]];`

and *average* merges results to:

`[['apple', 0.8], ['banana', 0.4], ['cherry', 0.4]].`

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

8.2.3 Merging Operators for Labels

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

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

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

8.3 Graph of Labels

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

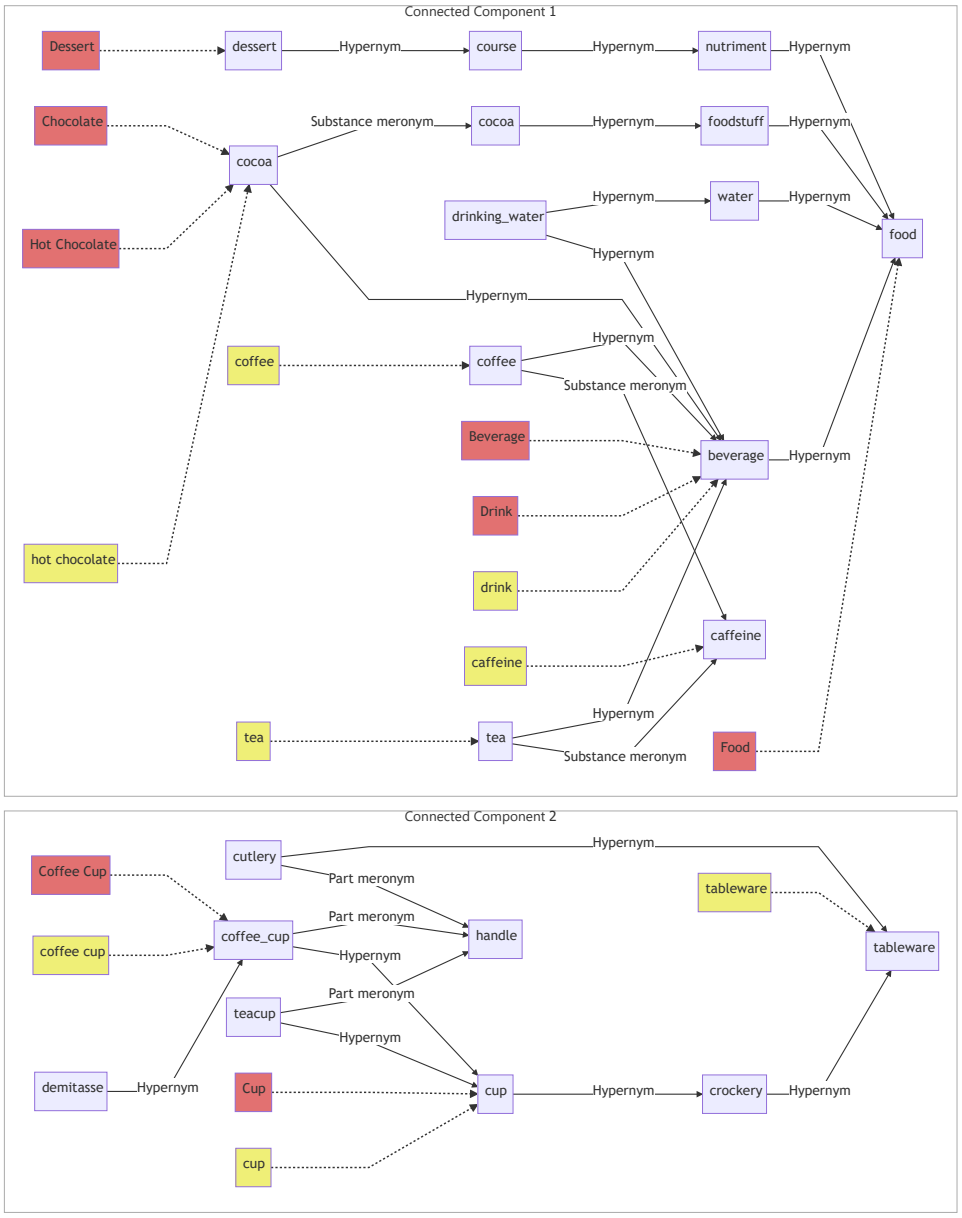


Figure 8.2: Graph of labels from two concrete endpoints (red and yellow) and their associated synsets related to both words (blue).

Table 8.1: Statistics for the number of labels, on average, per service identified.

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

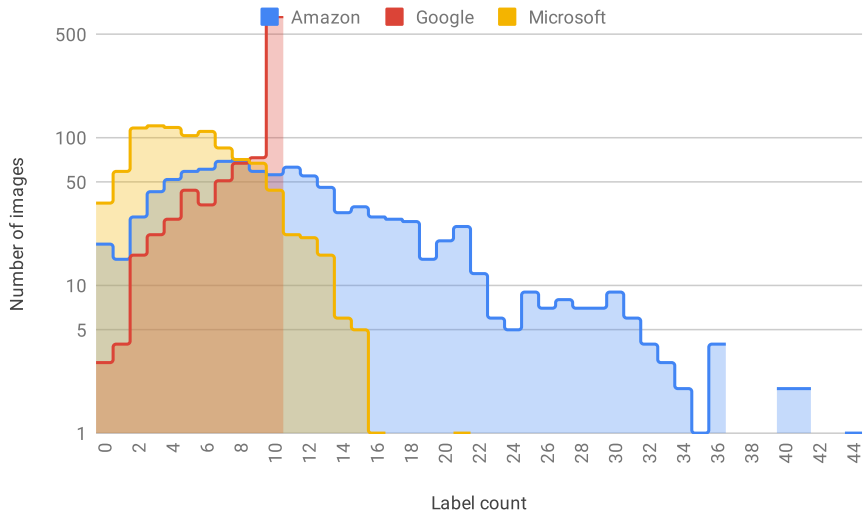


Figure 8.3: Number of labels responded from our input dataset to three concrete APIs assessed.

8.3.1 Labels and synsets

The number of labels depends on input images and concrete API endpoints used. **????** show how many labels are returned, on average per image, from Google Cloud Vision [**?**], Amazon Rekogition [**?**] and Azure Computer Vision [**?**] image recognition APIs. These statistics were calculated using 1,000 images from Open Images Dataset V4 [**?**] Image-Level Labels set.

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

8.3.2 Connected Components

A connected component (CC) is a subgraph in which there are paths between any two nodes. In graphs of labels and synsets, CCs are clusters of labels and synsets with similar semantic meaning. For instance, there are two CCs in **??**. CC 1 in **??** has `'beverage'`, `'dessert'`, `'chocolate'`, `'hot chocolate'`, `'drink'`, and `'food'` labels from the red first endpoint and `'coffee'`, `'hot chocolate'`,

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

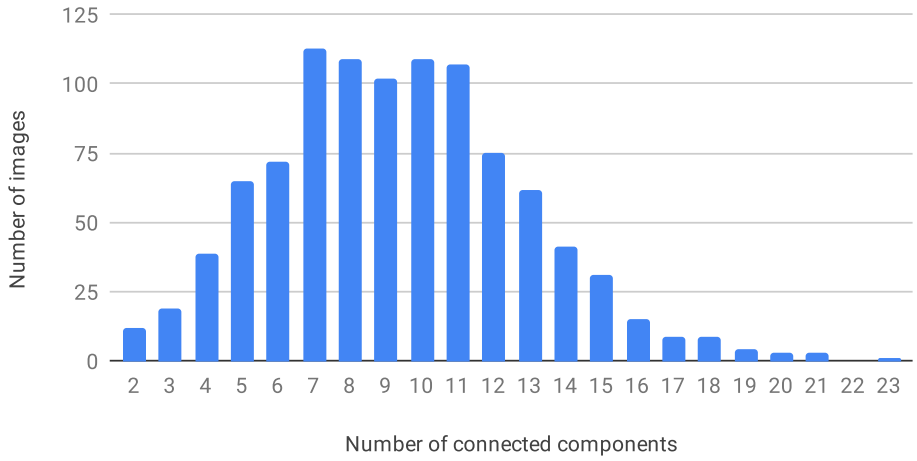


Figure 8.4: Number of connected components compared to the number of images.

`'drink'`, `'caffeine'`, and `'tea'` labels from the yellow second endpoint. Therefore, these labels are related to `'drink'`. On the other hand, CC 2 in **??** has `'cup'` and `'coffee cup'` labels from the first red endpoint and `'cup'`, `'coffee cup'`, and `'tableware'` labels from the yellow second endpoint. These labels are, therefore, related to `'cup'`.

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

8.4 API Results Merging Algorithm

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

8.4.1 Mapping Labels to Synsets

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

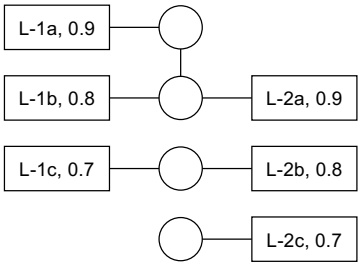


Figure 8.5: Allocation to connected components.

8.4.2 Deciding Total Number of Labels

The number of labels in responses from endpoints vary as described in **??**. The facade decides the number of merged labels using the numbers of labels from each endpoint. We formulate the following equation to calculate the number of labels:

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

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

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

The proposal uses $\lfloor \Sigma_i|R_i|/n \rfloor$ to conform to the necessary condition described in **??**.

8.4.3 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 **??**, 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 [**?**]. 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

Table 8.2: Allocation iteration 1.

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

Table 8.3: Allocation iteration 2.

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

Table 8.4: Allocation iteration 3.

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

Table 8.5: Allocation iteration 4.

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

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 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 **??**.

???????? are examples of allocation iterations. In **??**, the facade sorts scores separately for each endpoint. For instance, the first CC in **??** has scores of 0.9 and 0.8 from endpoint 1 and 0.9 from endpoint 2. All CCs have a non-empty score array or more, so the facade skips **??**. The facade then picks the highest scores for each endpoint and CC. CC 1 has the largest product of highest scores and receives an allocation. In **??**, the first CC removes every first score in its array as it received an allocation in **??**. In this iteration, the second CC has largest product of scores and receives an allocation. In **??**, the second CC removes every first score in its array. At **??**, all the three CCs have an empty array. The facade removes one empty array from each CC. In **??**, the first CC receives an allocation. The algorithm is applicable if total number of allocation is less than or equal to $\max_i(|R_i|)$ as scores are removed in **??**. The condition is a necessary condition.

8.4.4 Selecting Labels from Connected Components

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

8.4.5 Conformance to properties

?? defines four properties: identity, commutativity, reflexivity, and additivity. Our proposed method conforms to these properties:

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

8.5 Evaluation

8.5.1 Evaluation Method

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

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

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

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

Let R_A, R_G, and R_M be responses from Amazon Rekognition, Google Cloud Vision, and Microsoft's Azure Computer Vision, respectively. There are four sets of operands, i.e., (R_A, R_G), (R_G, R_M), (R_M, R_A), and (R_A, R_G, R_M). **??** shows the

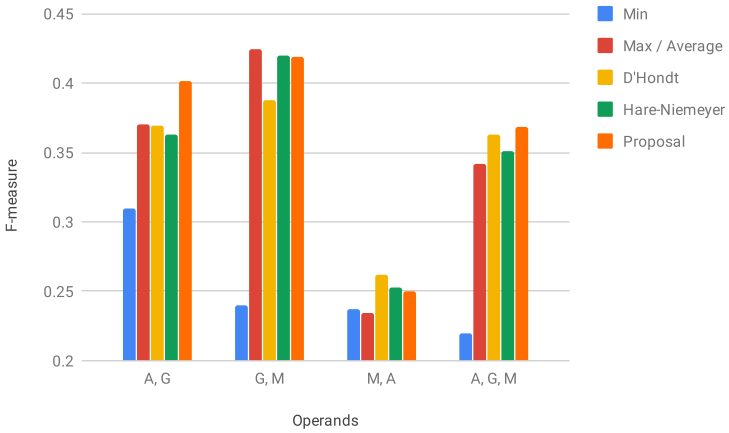


Figure 8.6: F-measure comparison.

evaluation of each operands set, **??** shows the averages of the four operands sets, and **??** shows the comparison of F-measure for each methods.

8.5.2 Naive Operators

Results of *min*, *max*, and *average* operators are shown in **??????**. The *min* operator is similar to *union* operator of set operation, and outputs all labels of operands. The precision of the *min* operator is always greater than any precision of operands, and the recall is always lesser than any precision of operands. *Max* and *average* operators are similar to *intersection* operator of set operations. Both operators output intersection of labels of operands and there is no clear relation to the precision and recall of operands. Since both operators have the same precision, recall, and F-measure, **??** groups them into one. The *average* operator performs well on the TP/FP ratio, where most of the same labels from multiple endpoints are TPs. In many cases of the four operand sets, all naive operators' F-measures are between F-measures of operands. None of naive operators therefore improve results by merging responses from multiple endpoints.

8.5.3 Traditional Proportional Representation Operators

There are many existing allocation algorithms in proportional representation, e.g., the **?** method [**?**]. These methods may be replacements of those in **??**. Other steps, i.e., **??????**, are the same as for our proposed technique. **??????** show the result of these traditional proportional representation algorithms. Averages of F-measures by traditional proportional representation operators are almost equal to that of the *max* and *average* operators. It is worth noting that merging *M* and *A* responses results in a better F-measure than each F-measure of *M* and *A* individually. As these are not biased to human-verified labels, situations in the real-world usage should, therefore, be similar to the case of *M* and *A*. Hence, RQ1 is true.

Table 8.6: Matching to human-verified labels.

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

Table 8.7: Evaluation results. A = Amazon Rekognition, G = Google Cloud Vision, M = Microsoft's Azure Computer Vision.

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

Table 8.8: Average of the evaluation result.

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

8.5.4 New Proposed Label Merge Technique

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

8.5.5 Performance

We used AWS EC2 m5.large instance (2 vCPUs, 2.5 GHz Intel Xeon, 8 GiB RAM); Amazon Linux 2 AMI (HVM), SSD Volume Type; Node.js 8.12.0. It takes 0.370 seconds to merge responses from three endpoints. Computational complexity of the algorithm in **??** is $O(n^2)$, where n is total number of labels in responses. (The estimation assumes that the number of endpoints is a constant.) Complexity of **??** in **??** is $O(n \log n)$, as the worst case is that all n labels are from one single endpoint and all n labels are in one CC. Complexity of **??** to **??** is $O(n^2)$, as the number of CCs is less than or equal to n and number of iterations are less than or equal to n. As **??** shows, the averaged total number of three endpoints is 25.58. Most of time for merging is consumed by looking up WordNet synsets (**??**). The API facade calls each APIs on actual endpoints in parallel. It takes about 5 seconds, which is much longer than 0.370 seconds taken for the merging of responses.

8.6 Conclusions and Future Work

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

Our idea of a proportional representation approach can be applied to other IWSs. If the response of such a service is list consisting of an entity and score, and if there is a way to group entities, a proposal algorithm can be applied. The opposite approach is to improve results by inferring labels. Our current approach picks some of the

labels returned by endpoints. IWSs are not only based on supervised ML—thus to cover a wide range of IWSs, it is necessary to classify and analyse each APIs and establish a method to improve results by merging. Currently graph structures of labels and synsets (**??**) are not considered when merging results. Propagating scores from labels could be used, losing the additivity property but improving results for users. There are many ways to propagate scores. For instance, setting propagation factors for each link type would improve merging and could be customised for users' preferences. It would be possible to generate an API facade automatically. APIs with the same functionality have same or similar signatures. Machine-readable API documentation, for instance, OpenAPI Specification, could help a generator to build an API facade.

CHAPTER 9

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

Abstract Increased popularity of 'intelligent' web services provides end-users with machine-learnt functionality at little effort to developers. However, these services require a decision threshold to be set which is dependent on problem-specific data. Developers lack a systematic approach for evaluating intelligent services and existing evaluation tools are predominantly targeted at data scientists for pre-development evaluation. This paper presents a workflow and supporting tool, Threshy, to help *software developers* select a decision threshold suited to their problem domain. Unlike existing tools, Threshy is designed to operate in multiple workflows including pre-development, pre-release, and support. Threshold configuration files exported by Threshy can be integrated into client applications and monitoring infrastructure. Demo: `https://bit.ly/2YKeYhE`.

9.1 Introduction

Machine learning algorithm adoption is increasing in modern software. End users routinely benefit from machine-learnt functionality through personalised recommendations [**?**], voice-user interfaces [**?**], and intelligent digital assistants [**?**]. The easy accessibility and availability of intelligent web services[1] is contributing to their adoption. These intelligent web services simplify the development of machine learning solutions as they (i) do not require specialised machine learning expertise to build and maintain, (ii) abstract away infrastructure related issues associated with machine learning [**?** **?**], and (iii) provide web APIs for ease of integration.

However, unlike traditional web services, the functionality of these *intelligent services* is dependent on a set of assumptions unique to machine learning [**?**]. These assumptions are based on the data used to train machine learning algorithms,

[†]This chapter is originally based on . Terminology has been updated to fit this thesis.

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

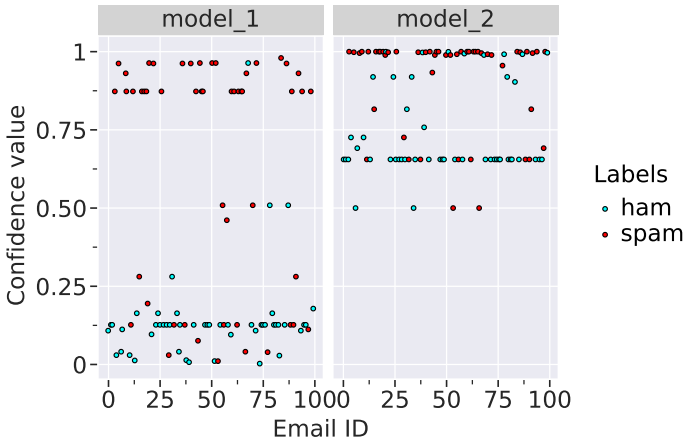


Figure 9.1: Predictions for 100 emails from two spam classifiers. Decision thresholds are classifier-dependent: a single threshold for both classifiers is *not* appropriate as ham emails are clustered at 0.12 (model_1) and at 0.65 (model_2). Developers must evaluate performance for *both* thresholds.

the choice of algorithm, and the choice of data processing steps—most of which are not documented. For developers, these assumptions mean that the performance characteristics of an intelligent service in any particular application problem domain is not fully knowable. Intelligent services represent this uncertainty through a confidence value associated with their predictions. Thus an evaluation procedure must be followed as a part of using an intelligent service for an application.

A typical evaluation process would involve a test data set (curated by the developers using the intelligent service) that is used to determine an appropriate threshold. Choice of a decision threshold is a critical element of the evaluation procedure [**?**]. This is especially true for classification problems such as detecting if an image contains cancer or identifying all of the topics in a document. Simple approaches to selecting a threshold are often insufficient, as highlighted in Google's machine learning course: *"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****."*[2] As an example consider the predictions from two email spam classifiers shown in **??**. The predicted safe emails, 'ham', are in two separate clusters (a simple threshold set to approx. 0.2 for model 1 and 0.65 for model 2, indicating that different decision thresholds may be required depending on the classifier. Also note that some emails have been misclassified; how many depends on the choice of decision threshold. An appropriate threshold considers factors outside algorithmic performance, such as financial cost and impact of wrong decisions. To select an appropriate decision threshold, developers using intelligent services need approaches to reason about and consider trade-offs between competing *cost factors*. These include impact, financial costs, and maintenance implications. Without considering these trade-offs, sub-optimal decision thresholds will be selected.

The standard approach for tuning thresholds in classification problems involve

[2]See `https://bit.ly/36oMgWb`.

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

In this paper we present ***Threshy***[3], a tool to assist developers in selecting decision thresholds when using intelligent services. The motivation for developing Threshy arose from our consultancy work with industry. Unlike existing tooling (see **??**), **Threshy serves as a means to up-skill and educate software engineers in selecting machine-learnt decision thresholds**, for example, on aspects such as confusion matrices. Threshy provides a visually interactive interface for developers to fine-tune thresholds and explore trade-offs of prediction hits/misses. This exposes the need for optimisation of thresholds, which is dependent on particular use cases.

Threshy improves developer productivity through automation of the threshold selection process by leveraging an optimisation algorithm to propose thresholds. The algorithm considers different cost factors providing developers with summary information so they can make more informed trade-offs. Developers also benefit from the workflow implemented in Threshy by providing a reproducible procedure for testing and tuning thresholds for any category of classification problem (binary, multi-class, and multi-label). Threshy has also been designed to work for different input data types including images, text and categorical values. The output, is a text file and can be integrated into client applications ensuring that the thresholds can be updated without code changes (if needed), and continuously monitored in a production setting.

9.2 Motivating Example

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

Nina uses a two-stage pipeline consisting of a multi-class and a binary classification model. She has decided to evaluate the viability of cloud based intelligent

[3]Threshy is available for use at `http://bit.ly/a2i2threshy`.

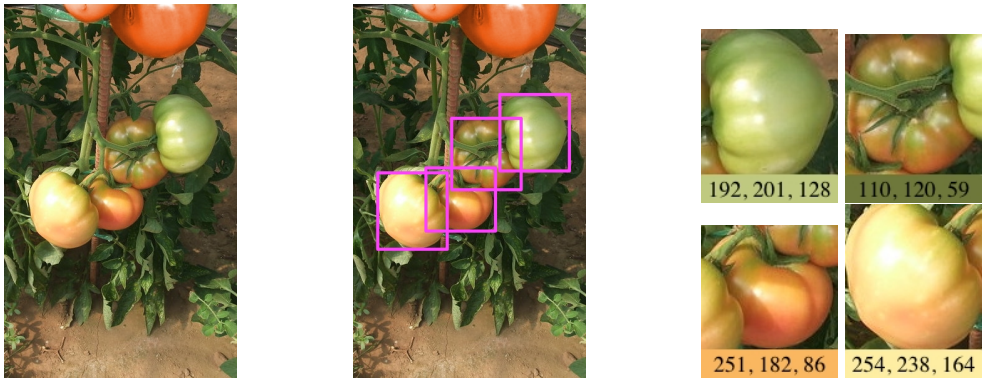


Figure 9.2: Pipeline of Nina's harvesting robot. *Left:* Photo from harvesting robot's webcam. *Centre:* Classification detecting different types of tomatoes. *Right:* Binary classification for ripeness (ripe/unripe) based on (R, G, B values).

services and use them if operationally effective. **??** illustrates an example of the the pipeline as listed below:

1. **Classify tomato 'type'.** This stage uses an object localisation service to detect all tomato-like objects in the frame and classifies each tomato into one of the following labels: `['roma','cherry','plum','green','yellow']`.

2. **Assess tomato 'ripeness'.** This stage uses a crop of the localised tomatoes from the original frame to assess the crop's colour properties (i.e., average colour must have R > 200 and G < 240). This produces a binary classification to deduce whether the tomato is ripe or not.

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

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

Nina combined the predictions, their respective confidence values, and Lucy's labelled ground truths into a CSV file which was then uploaded to Threshy. Nina asked Lucy to assist in setting relevant costs for correct predictions and false predictions. Threshy then recommended a choice of decision threshold which Nina then fine tuned while considering the performance and cost implications.

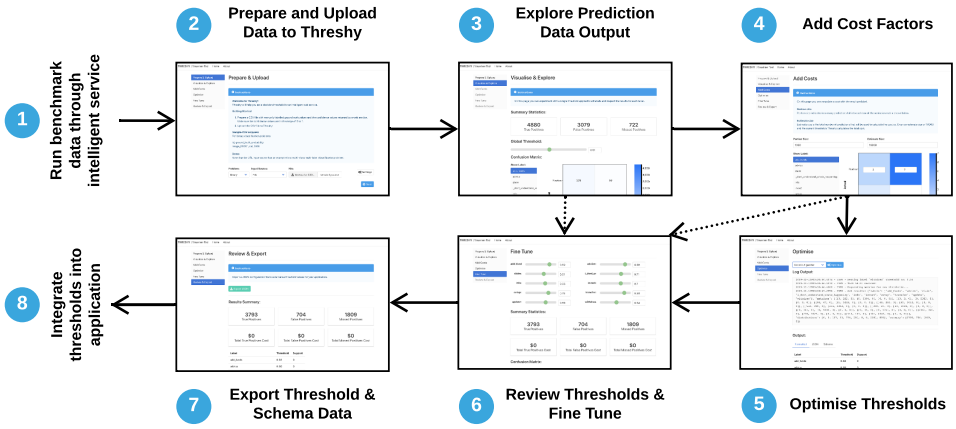


Figure 9.3: UI workflow for interacting with Threshy to optimise the thresholds for classification problem.

9.3 Threshy

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

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

Setting thresholds in Threshy is an eight step process as shown in **??**. Software engineers ① run a benchmark dataset through the machine-learnt component to create a CSV file with true labels and predicted labels along with the predicted confidence values. The CSV file is then ② uploaded for initial exploration where engineers can ③ experiment with modifying a single global threshold for the dataset. Developers may choose to exit at this point (as indicated by dotted arrows in **??**). Optionally, the engineer ④ defines costs for missed predictions followed by selecting optimisation settings. The optional optimisation step of Threshy ⑤ considers the performance and costs when deriving the thresholds. Finally, the engineer can ⑥ review and fine tune the calculated thresholds, associated costs, and ⑦ download

generated threshold meta-data to be ⑧ integrated into their application.

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

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

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

9.4 Related work

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

However, an *automated* attempt to calibrate decision threshold boundaries is not considered, and is largely pitched at a non-software engineering audience. A more recent study touches on this in model management for large-scale adversarial instances in Google's advertising system [**?**], however this is only a single component within the entire architecture, and is not a tool that is useful for developer's in varying contexts. Unlike this study, our work presents a 'plug-and-play' style calibration

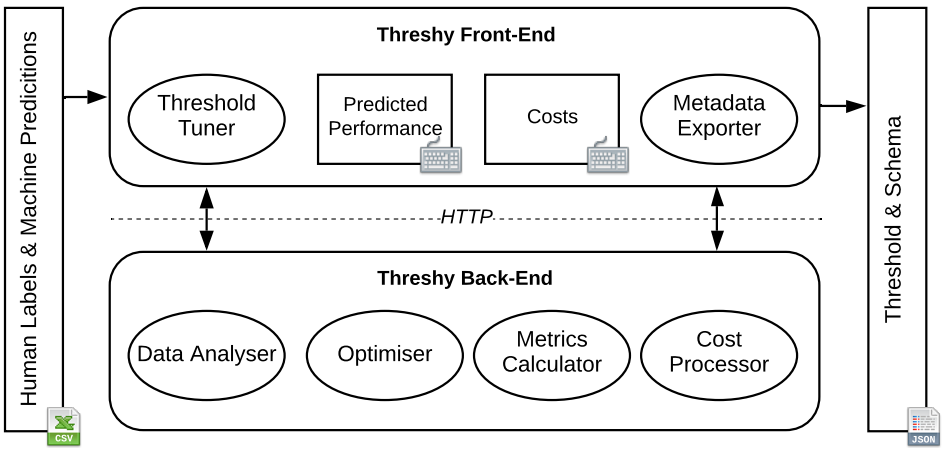


Figure 9.4: Architecture of Threshy.

method where any context/domain can have thresholds automatically calibrated (in-context) *and* optimised for engineers; Threshy's architecture and design facilitates operating in a headless mode enabling use in monitoring and support workflows.

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 [**?** **?**].) 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 **??**) to reflect the nature of real-world development. Google's *What-If Tool* [**?**] 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 [**?**] provides an interface to test hypotheticals (although only on Generalized Additive Models) and their *ModelTracker* tool [**?**] collates summary statistics on a set of sample data to enable rich visualisation of model behaviour and access to key performance metrics.

However, these tools are largely focused toward pre-development model evaluation and are not designed for the software engineering workflow. They are also targeted to data scientists and not engineers, and certain tools are tied to specific machine learning frameworks (e.g., What-If and Tensorflow). Our work attempts to bridge these gaps through a structured workflow with an automated tool targeted to software developers. We also consider the need to have a consistent tool that works across development, test, and production environments.

9.5 Conclusions & Future Work

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

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

CHAPTER 10

FSE Paper[†]

[†]This chapter is originally based on . Terminology has been updated to fit this thesis.

Part III

Postface

CHAPTER 11

Conclusions & Future Work

List of Online Artefacts

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

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

Part IV

Appendices

APPENDIX A

Additional Materials

A.1 The Development, Documentation and Usage of Web APIs

The development of web application programming interfaces (APIs) (commonly referred to as a *web service*) and web APIs 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 [**?**]. This framework used the synchronous communication paradigm remote procedure calls (RPCs) first introduced by **?**] that allows procedures to be called in a remote address space as if it were local. Its communication paradigm, DCE/RPC [**?**], 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 [**?**]). 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 (WS) applications, a realisation of the service-oriented architecture (SOA) [**?**] 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 **??**) 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 [**?**]. 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 [**?** **?**]). These attributes are included within SOAP headers; thus, QoS aspects are independent from the transport layer and instead exist at the application layer [**?**]. 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 [**?** **?**] 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 (**????**).

Listing A.1: A SOAP HTTP POST consumer request to retrieve customer record #43456 from a web service provider. Source: [**?**].

Figure A.1: Worldwide search interest for SOAP (blue) and REST (red) since 2004. Source: Google Trends.

```
POST /customers HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8

<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <m:GetCustomer
      xmlns:m="http://www.example.org/customers">
      <m:CustomerId>43456</m:CustomerId>
    </m:GetCustomer>
  </soap:Body>
</soap:Envelope>
```

Listing A.2: The SOAP HTTP service provider response for **??**. Source: [**?**].

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8

<?xml version='1.0' ?>
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
  <env:Body>
    <m:GetCustomerResponse
      xmlns:m="http://www.example.org/customers">
      <m:Customer>Foobar Quux, inc</m:Customer>
    </m:GetCustomerResponse>
  </env:Body>
</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 [**?**]. That is, the HTTP is considered as a transport

protocol solely. In 2000, **?**] 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. **?** 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 [**?**].
- **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 **????** but in a RESTful architecture (**????**) and it is clear why this style has grown more popular with developers (as we highlight in **??**). 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) [**?**]). Therefore, no RESTful service uses a standardised response document or invocation syntax. While there are proposals, such as WADL [**?**], RAML[1], API Blueprint[2], and the OpenAPI[3] specification (initially based on Swagger[4]), 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 **??**, but using REST. Source: [**?**].

```
1 GET /customers/43456 HTTP/1.1
2 Host: www.example.org
```

Listing A.4: The REST HTTP service provider response for **??**.

```
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.

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

Category	Sample vendors and products	Typical use cases
Embedded AI Expert assistants leverage AI technology embedded in platforms and solutions.	• Amazon: Alexa • Apple: Siri • Facebook: Messenger • Google: Google Assistant (and more) • Microsoft: Cortana • Salesforce: MetaMind (acquisition)	• Personal assistants for search, simple inquiry, and growing as expert assistance (composed problems, not just search) • Available on mobile platforms, devices, the internet of things • Voice, image recognition, various levels of NLP sophistication • Bots, agents
AI point solutions Point solutions provide specialized capabilities for NLP, vision, speech, and reasoning.	• 24[7]: 24[7] • Admantx: Admantx • Affectiva: Affdex • Assist: AssistDigital • Automated Insights: Wordsmith • Beyond Verbal: Beyond Verbal • Expert System: Cogito • HPE: Haven OnDemand • IBM: Watson Analytics, Explorer, Advisor • Narrative Science: Quill • Nuance: Dragon • Salesforce: MetaMind (acquisition) • Wise.io: Wise Support	• Semantic text, facial/visual recognition, voice intonation, intelligent narratives • Various levels of NLP from brief text messaging, chat/conversational messaging, full complex text understanding • Machine learning, predictive analytics, text analytics/mining • Knowledge management and search • Expert advisors, reasoning tools • Customer service, support • APIs
AI platforms Platforms that offer various AI tech, including (deep) machine learning, as tools, APIs, or services to build solutions.	• CognitiveScale: Engage, Amplify • Digital Reasoning: Synthesys • Google: Google Cloud Machine Learning • IBM: Watson Developers, Watson Knowledge Studio • Intel: Saffron Natural Intelligence • IPsoft: Amelia, Apollo, IP Center • Microsoft: Cortana Intelligence Suite • Nuance: 360 platform • Salesforce: Einstein • Wipro: Holmes	• APIs, cloud services, on-premises for developers to build AI solutions • Insights/advice building • Rule-based reasoning • Vertical domain advisors (e.g., fraud detection in banking, financial advisors, healthcare) • Cognitive services and bots
Deep learning Platforms, advanced projects, and algorithms for deep learning.	• Amazon: FireFly • Google: TensorFlow/DeepMind • LoopAI Labs: LoopAI • Numenta: Grok • Vicarious: Vicarious	• Deep learning neural networks for categorization, clustering, search, image recognition, NLP, and more • Location pattern recognition • Brain neocortex simulation

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

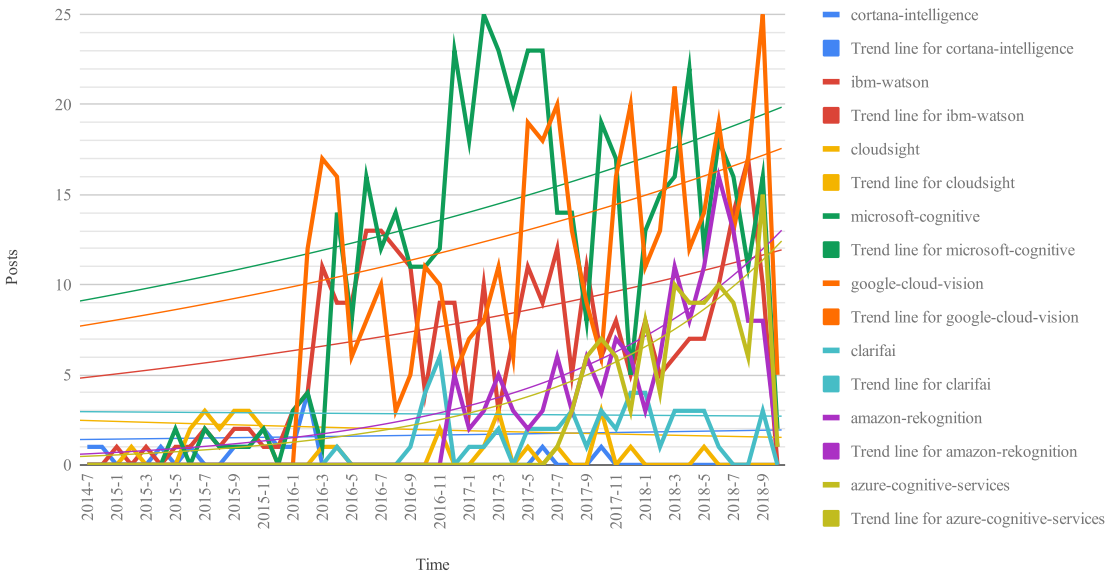


Figure A.4: Questions asked by software engineering researchers (column 2) that can be answered by field study techniques. (From [**?**].)

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

APPENDIX B

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.*	Yes

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. *(If the same, write "as above")*	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Provenance in Large-Scale Artificial Intelligence Solutions
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.

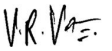
If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

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

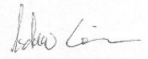
Author 3 John Grundy

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 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.*	Yes

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. *(If the same, write "as above")*	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Provenance in Large-Scale Artificial Intelligence Solutions
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication.	See page 2

I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.

Signed:  Dated: 22 July 2019

4. Description of all author contributions

Name and affiliation of author 1	Alex Cummaudo Applied Artificial Intelligence Institute Deakin University
Contribution of author 1	Alex Cummaudo devised the conception of this project and the intended objectives and hypotheses. Additionally, he designed a detailed methodology, conducted data collection with a custom tool he wrote himself and performed analysis. He drafted the manuscript and made further revisions, modifications and prepared the camera ready version for publication in the conference proceedings.
Name and affiliation of author 2	Rajesh Vasa Applied Artificial Intelligence Institute Deakin University
Contribution of author 2	Rajesh Vasa contributed to the initial conception of this project by providing high-level guidance over overview of what the project and its experiments should comprise of. Rajesh also contributed to detailed revisions of the initial manuscripts, and assisted in advising Alex Cummaudo on improved analytical insight into the collected results. Rajesh Vasa also assisted in shaping the paper to specifically target the conference audience. Rajesh Vasa is the primary supervisor of Alex Cummaudo.
Name and affiliation of author 3	John Grundy Faculty of Information Technology Monash University
Contribution of author 3	John Grundy provided high-level oversight of the project. He contributed to detailed reviews of the methodology and manuscript. John Grundy is the external supervisor of Alex Cummaudo.

5. Author declarations

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

i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,

ii. that there are no other authors according to these criteria,

iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

iv. that the data on which these findings are based are stored as set out in Section 7 below.

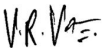
If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Author 1 Alex Cummaudo

Signed: 
Dated: 22 July 2019

Author 2 Rajesh Vasa

Signed: 
Dated: 22 July 2019

Author 3 John Grundy

Signed: 
Dated: 22 July 2019

6. Other contributor declarations

There are no other contributors for this publication to declare.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Comma separated values (CSV), Portable Document Format (PDF)
Storage location	Deakin University Research Data Store (RDS) Location: RDS29448-Alex-Cummaudo-PhD/results/esem19

8. Additional notices

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

Deakin University Authorship Procedure
Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	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 Ohtake
School/Institute/Division if at Deakin Organisation and address if non-Deakin	Faculty of Science, Engineering and Built Environment
Email or phone	tomohiro.otake@deakin.edu.au

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis? *If Yes, please complete Section 3* *If No, go straight to Section 4.*	Yes

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. *(If the same, write "as above")*	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Provenance in Large-Scale Artificial Intelligence Solutions
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,

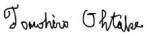
iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

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 Ohtake

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.

APPENDIX C

Ethics Clearance

Rajesh Vasa and Alex Cummaudo
Applied Artificial Intelligence Institute (A^2I^2)
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

Teresa Treffry
Secretary, Human Ethics Advisory Group (HEAG)
Faculty of Science Engineering & Built Environment

Faculty of Science Engineering & Built Environment Human Ethics Advisory Group
Geelong Waurn Ponds Campus, Geelong, VIC 3220
Tel 03 522 72270 Fax 03 522 72028 sciethic@deakin.edu.au www.deakin.edu.au

Rajesh Vasa, Mohamed Abdelrazek, Andrew Cain, Scott Barnett, Alex Cummaudo
Applied Artificial Intelligence Institute (A^2I^2) (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

APPENDIX D

Primary Sources from Systematic Mapping Study

References

[1] M. P. Robillard, "What makes APIs hard to learn? Answers from developers," *IEEE Software*, vol. 26, no. 6, pp. 27–34, 2009.

[2] M. P. Robillard and R. Deline, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[3] A. J. Ko and Y. Riche, "The role of conceptual knowledge in API usability," in *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2011, pp. 173–176.

[4] 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." *SIGDOC*, 2002.

[5] R. Watson, M. Mark Stamnes, J. Jeannot-Schroeder, and J. H. Spyridakis, "API documentation and software community values," in *the 31st ACM international conference*. New York, New York, USA: ACM Press, 2013, pp. 165–10.

[6] 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 *International Symposium on End User Development*. Springer, 2009, pp. 86–105.

[7] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, "Software Documentation Issues Unveiled," *ICSE*, vol. 2, no. 3, pp. 127–139, 2019.

[8] S. Haselbock, R. Weinreich, G. Buchgeher, and T. Kriechbaum, "Microservice Design Space Analysis and Decision Documentation: A Case Study on API Management," *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 1–8, Nov. 2018.

[9] S. Inzunza, R. Juárez-Ramírez, and S. Jiménez, "API Documentation," in *Trends and Advances in Information Systems and Technologies*. Cham: Springer, Cham, Mar. 2018, pp. 229–239.

[10] 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, Aug. 2017.

[11] 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," *Computer Supported Cooperative Work (CSCW)*, vol. 27, no. 3-6, pp. 767–802, May 2018.

[12] A. Head, C. Sadowski, E. Murphy-Hill, and A. Knight, *When not to comment: questions and tradeoffs with API documentation for C++ projects*, ser. questions and tradeoffs with API documentation for C++ projects. New York, New York, USA: ACM, May 2018.

[13] L. Aversano, D. Guardabascio, and M. Tortorella, "Analysis of the Documentation of ERP Software Projects," *Procedia Computer Science*, vol. 121, pp. 423–430, Jan. 2017.

[14] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong, "On-demand Developer Documentation," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 479–483.

[15] R. B. Watson, "Development and application of a heuristic to assess trends in API documentation." *SIGDOC*, 2012.

[16] W. Maalej and M. P. Robillard, "Patterns of Knowledge in API Reference Documentation." *IEEE Trans. Software Eng.*, 2013.

[17] D. L. Parnas and S. A. Vilkomir, "Precise Documentation of Critical Software," in *10th IEEE High Assurance Systems Engineering Symposium (HASE'07)*. IEEE, Sep. 2007, pp. 237–244.

[18] C. Bottomley, "What part writer? What part programmer? A survey of practices and knowledge used in programmer writing," *IPCC 2005. Proceedings. International Professional Communication Conference, 2005.*, pp. 802–812, Jan. 2005.

[19] A. Taulavuori, E. Niemelä, and P. Kallio, "Component documentation—a key issue in software product lines," *Information and Software Technology*, vol. 46, no. 8, pp. 535–546, Jun. 2004.

[20] J. Kotula, "Using Patterns To Create Component Documentation." *IEEE Software*, 1998.

[21] S. G. McLellan, A. W. Roesler, J. T. Tempest, and C. Spinuzzi, "Building More Usable APIs." *IEEE Software*, 1998.