Taming the evolving black box: A new architectural model for intelligent 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

December 18, 2019

Abstract

Application developers are eager to integrate 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: A new architectural model for intelligent 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)
December 18, 2019

Dedicated to all my teachers.

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Contents

List of Figures

List of Tables

List of Listings

Part I

Preface

CHAPTER 1

Introduction

Within the last half-decade, we have seen an explosion of cloud-based services typically marketed under an AI banner. Vendors are rapidly pushing out AI-based solutions, technologies and products that encapsulate half a century worth of machine-learning research: a 2016 report by market research company Forrester captured such growth into four key areas [141] as replicated in Figure A.2. 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.

A contrast is how these systems shift away from the traditional software engineering paradigm that application developers are comfortable with. Typical systems built by application developers are *rule-driven* (or algorithm-driven), deterministically human engineered using source code to drive each step behind the application. These rule-driven systems typically consume, utilise, and integrate libraries and frameworks, IDEs and other tooling, and cloud-based services such as AWS [270]. The AI-first software is, however, not rule-driven but *data-driven*, fuelled with large datasets used to train ML prediction algorithms and classifiers that present a typically probabilistic and nondeterministic behaviour.

So, how does the application developer approach her AI-first application?

1. She writes her own ML classifier from scratch and trains it from her own curated dataset. This approach is laborious in time and demands formal training in ML and mathematical knowledge, but the tradeoff is that she has full autonomy over the models she creates.
2. She 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 still requires an understanding of how to 'glue' the ML framework code[1]

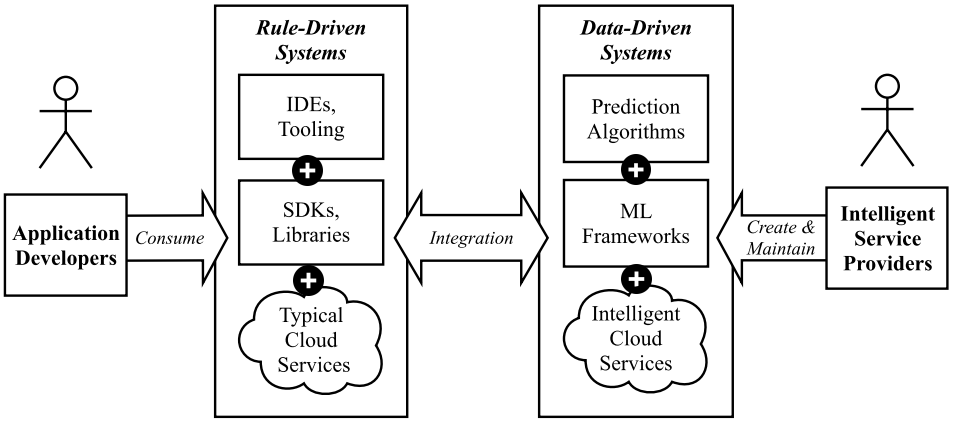
[1]Thus introducing a verbose list of ML terminology to her already-required developer vocabulary. See a list of 328 terms provided by Google here: `https://developers.google.com/`

with her own application code.

3. She uploads her data to a pre-existing cloud-based service: a *Friendly* service. She doesn't need to know anything behind the underlying 'intelligence' and how it works, is fast to integrate into her application and all abstracted behind a web-based API call.

At first sight, she perceives the data-driven cloud service as 'just another' cloud service offered in her toolchain. Her perception is that just because this is another cloud service, it should act and behave as any other typical service would. But internally, she isn't aware that it this particular cloud service isn't a typical one; the data-driven service does not adapt to her rule-driven application in the way that she thinks (Figure 1.1). This is because the nature of typical cloud systems and data-driven ones are not exactly the same (Table 1.1).

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.



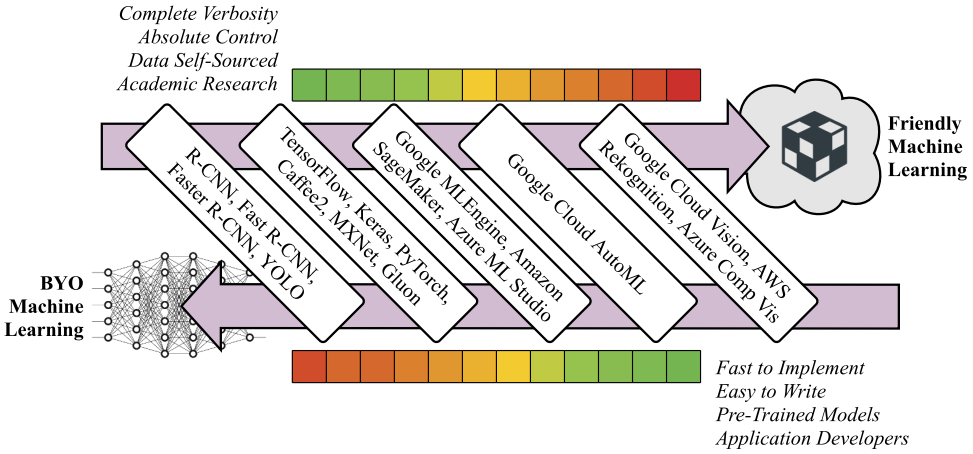
This 'range' of AI-first integration techniques partially reflects Google AI's[2] *Spectrum* [3, 128, 171], which encompasses the variety of skill, effort, users and types of outputs of integration techniques. On one extreme is the research of developing algorithms to achieve intelligence, produced chiefly in academia—coined as BYOML [2, 3, 171]. On the other, such intelligence becomes heavily abstracted as easy-to-use APIs, targeted mainly towards developers as 'friendly' ML. In the middle lies a broad mix of combining both cloud and locally-hosted solutions (with varying levels of automation to assist in development) that turn custom datasets into some form of predictive intelligence.

machine-learning/glossary/.

[2]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 can be seen through Google's 2018 rebranding of *Google Research* to *Google AI* [261] or how AI is leveraged *at scale* within Facebook's infrastructure and platforms to serve its users with an AI-first attitude [175].

All techniques in this spectrum are data-driven, and we illustrate their slightly varied characteristics further in Table 1.2 and examples of the computer vision spectrum in Figure 1.2.

Figure 1.2: Examples within the machine learning spectrum of computer vision. Benefits and drawbacks of each end of the spectrum are indicated with the colour scales.



In this study, we advocate that the (i) integration, (ii) documentation, and (iii) quality attributes of data-driven cloud services juxtaposes the rule-driven nature of end-applications as these intelligent cloud services are vastly different to their traditional counterparts, and great care must therefore be considered by application developers.

These cloud services have begun to gain traction within developer circles: Figure A.3 shows the increasing trend of posts since 2014 on Stack Overflow that categorise popular computer vision cloud APIs.[3] In academia, these 'off-the-shelf' and pre-packaged ML solutions present a varied nomenclature such as *Cognitive Applications* and *Machine Learning Services* [99] or *Machine Learning as a Service* [192]. Some services provide the infrastructure to rapidly begin training from custom datasets (Google's AutoML[4] is one such example) while others provide pre-trained datasets 'ready-for-use' in production without the need to train data. We refer to these latter services under the broader term *CIS*, and diagrammatically express their usage within Figure 1.3.

The general workflow of a CIS is simple: a developer accesses a CIS component via REST/SOAP API(s). For their given input, they receive an intelligent-like response typically serialised as JSON/XML. We note the intelligence component masks its 'intelligence' through a black-box: in recent years, there is a rise in providing human-level intelligence via crowdsourcing Internet marketplaces such as Amazon Mechanical Turk [**?**] or ScaleAPI [**?**]. Thus, a CIS may be powered by

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

[4]`https://cloud.google.com/automl/` last accessed 7 December 2018.

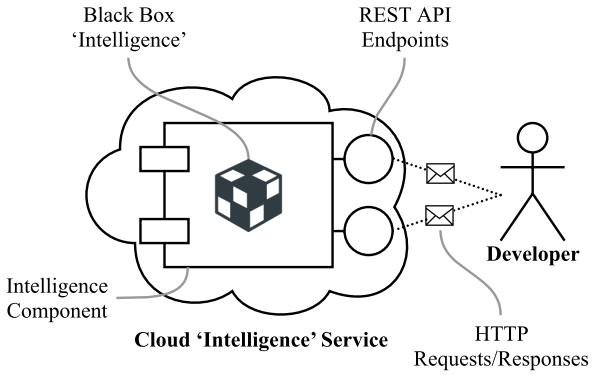
Table 1.1: Differing characteristics of intelligent and typical cloud services.

Intelligent Cloud Services	Typical Cloud Services
Probabilistic	Deterministic
Machine Learnt	Human Engineered
Data-Driven	Rule-Driven
Black-Box	Mostly Transparent

Table 1.2: Comparison of the machine learning spectrum.

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

Figure 1.3: Overview of Cloud Intelligence Services.



varying degrees of intelligence: human intelligence, machine learning, data mining or even intelligence by brute-force.

While there are different types of CISs evident (such as OCR transcription, text-to-speech and speech-to-text, object categorisation, object comparison, natural language processing etc.), we scope the work investigated in this study to CV-CISs [268**? ? ?** –277]. The ubiquity of CVCISs is exemplified through evermore growing applications that use these APIs: aiding the vision-impaired [49, 190], accounting [144], data analytics [105], and student education [55]. Moreover, we refer to its growing adoption in developer circles within Figure A.3.

1.1 Motivation: Current Developer Mindsets'

Figure A.3 shows an increasing trend to the adoption and discussion of CISs with developers. These services are accessible through APIs and consist of an 'intelligence' black box (Figure 1.3). When a term 'black box' is used, the input (or stimulus) is transformed to its outputs (or response) without any understanding of the internal architecture by which this transformation occurs, a theory arising from the electronic sciences and adapted to wider applications since the 1950s–60s [9, 31] to describe "systems whose internal mechanisms are not fully open to inspection" [9].

In the world of machine learning and data mining, where we develop algorithms to make predictions in our datasets or discover patterns within them, these black boxes are inherently probabilistic and stochastic; there is little room for certainty in these results as such insight is purely statistical and associational [179] against its training dataset. As an example, a CVCIS returns the *probability* that a particular object (the response) exists in the raw pixels (the stimulus), 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. Developers (at present) do not need to treat their programs in any such stochastic way given their rule-driven mindset that computers will always make

certain outcomes.

There are thus therefore three key factors to consider when implementing, testing and developing with a CIS: (i) the API usability, (ii) the nature of nondeterministic and probabilistic systems, and (iii) how both impact on software quality.

1.1.1 The Impact on Software Quality

Do traditional techniques for documenting deterministic APIs also apply to non-deterministic systems? As APIs reflect a set of design choices made by their providers intended for use by the developer, does the mindset between the machine learning architect and the novice programmer match? Evaluations of API usability advocate for the accuracy, consistency and completeness of APIs and their documentation [184, 198] written by providers, while providers should consider mismatches between the developer's conceptual knowledge of the API its implementation [122]. However, consistency cannot be guaranteed in probabilistic systems, and the conceptual knowledge of such systems are still treated like black boxes. It is therefore imperative that CIS providers consider the impact of their API usability; if not, poor API usability hinders on the internal quality of development practices, slowing developers down to produce the software they need to create.

Moreover, CIS APIs are inherently non-deterministic in nature, but developers are still taught with the deterministic mindset that all API calls are the same. Simple arithmetic representations (e.g., 2 + 2 = 4) will *always* result in 4; but a multi-layer perceptron neural network performing similar arithmetic representation [20] gives the probability where the target output (*exactly* 4) and the output inferred (*possibly* 4) matches as a percentage (or as an error where it does not match). That is, instead of an exact output, there is instead a *probabilistic* result: 2 + 2 *may* equal 4 with a confidence of n. External quality must therefore be considered in the outcome of these systems, such as in the case of thresholding values, to consider whether or not the inference has a high enough confidence to justify its result to end-users.

In order to fully understand this problem, there are multiple dimensions one must consider: the impact of software quality; the fact that these systems underneath are probabilistic and are stochastic; the cognitive biases of determinism in developers; the issue of consistency in API usage. While existing literature does extensively explore software quality and API usability, these studies have only had emphasis on deterministic systems and thus little work to date has investigated such factors on probabilistic systems that make up the core of CVCISs. We explore more of these facets in the motivating scenarios below.

1.1.2 Motivating Scenarios

The market for intelligent services is increasing (Figure A.2) and as is developer uptake and enthusiasm in the software engineering community (Figure A.3). We investigate the impact of the mismatch between the developer's mindset and the service provider's mindset as little work has been presented in literature. How do developers work with a CIS, how usable are these cloud APIs, and how well

do developers understand the non-deterministic and stochastic nature of a services backed by machine-learnt models?

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 CDSS that uses patient scans to recommend to surgeons if the patient should be sent to surgery.

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 as they are taken. 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 as well as common objects (e.g., all photos with his Border Collie dog, all photos taken on a beach on a sunny day). His app will then share all of this analysed intelligence of his photos with his friends on a social-media-like platform, where his friends can search and view the photos.

Rather than building a computer vision engine from scratch, which would take far too much time and effort, Tom thinks he can achieve this using one of the common CVCISs. 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 CVCIS 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 rightfully expects it to be consistent with the Android Java equivalent `"doggy".substring(0, 2)`. Consistent, here, means two things: (i) that `'dog'` will *always* be returned every time he invokes the method in either language (i.e., a static response); and (ii) that `'dog'` will *always* be returned regardless of what programming language or string library is used, given the deterministic nature of the 'substring' construct (i.e., results for substring are API-agnostic).

More concretely, in Table 1.3, we illustrate how three CVCIS providers (anonymised) fail to provide similar consistency to that of the substring example above. If Tom uploads a photo of a border collie[5] 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 also vary 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 even means. Their current tautological nature of the definition of these changing confidence values (as presented in the API documentation) provides no insight for Tom to understand why there was a change in confidence,

[5]The image used for these results can be found at `https://www.akc.org/dog-breeds/border-collie/`.

Table 1.3: First six responses of image analysis for a Border Collie sent to three computer vision CIS APIs 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

which we show in Table 1.4, unless he *knows* that the underlying models change with them. Thus, the deterministic problem of a substring compared to the nondeterministic nature of the CIS is not so simple to comprehend unless he is explicitly told.

To make an assessment of these APIs, he tries his best to read through the documentation of some CVCIS APIs, but he has no guiding framework to help him choose the right one. Some of the questions that come to mind include:

- What does confidence mean?
- Are these APIs consistent in how they respond?
- Will he need a combination of multiple CVCIS APIs to solve this task?
- 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 can pick up, and what labels it can't?
- How does it describe his photos and detect the faces?
- How can he interpret the results if he disagrees with it to help improve his app?
- Does he understand that the API uses a machine learnt model? Does he know what a ML model even is?
- If so, does he know when the models update? What is the release cycle?

Table 1.4: Tautological definitions of 'confidence' found in the API documentation of three common CVCIS 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)."* [262] *"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]."* [263] *"The overall score of the result. Range [0, 1]"* [263]
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."* [264] *"The level of confidence the service has in the caption."* [265]
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."* [266] *"[Provider C] also provide[s] a percentage score for how much confidence [Provider C] has in the accuracy of each detected label."* [267]

Dazzled by this, he does some brief reading on Wikipedia but is confused by the immense technical detail to take in. He would like some form of guiding communication framework to assist him and in software engineering terms aligned to his background.

Although Tom generally anticipates some imperfections, he has no prior benchmark to guide him on what to expect. He understands that the app is not always going to be perfect: perhaps some photos of his dog may be missed because the dog is in the background and not the foreground, or his friends can't find the photos of their recent trip to the beach because it wasn't sunny enough for the beach to be recognised. These imperfections appear to be low-risk, but may become socially awkward when in use; for instance, if some of Tom's friends have low self-esteem and use the app, they may be sensitive to being misidentified or even mislabelled. Privacy issues also come into play especially if certain friends have only 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, then Tom's partner's privacy is at risk.

Motivating Scenario II: Cancer Detection CDSS

Recent works in the oncology domain have used deep-learning CNNs to detect ROIs in image scans of tissue (e.g., [64, 89, 140]), 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. Some studies have suggested that practitioner over-reliance may erode independent decision-making [39, 107]; therefore the risks in developing CDSSs powered by intelligent services become paramount.

In Figure 1.4 we present a context diagram for a fictional CDSS named *CancerAssist*. CancerAssist is used by a team of busy pathologists who review patient lymph node scans and discuss and recommend, on consensus, if the patient should or should not be sent to surgery. When consensus is made, the lead pathologist enters the verdict into CancerAssist—running passively in the background—to ensure no oversight has been made 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 override the output of CancerAssist, this helps to reinforce CancerAssist's internal model as a HITL learning process.

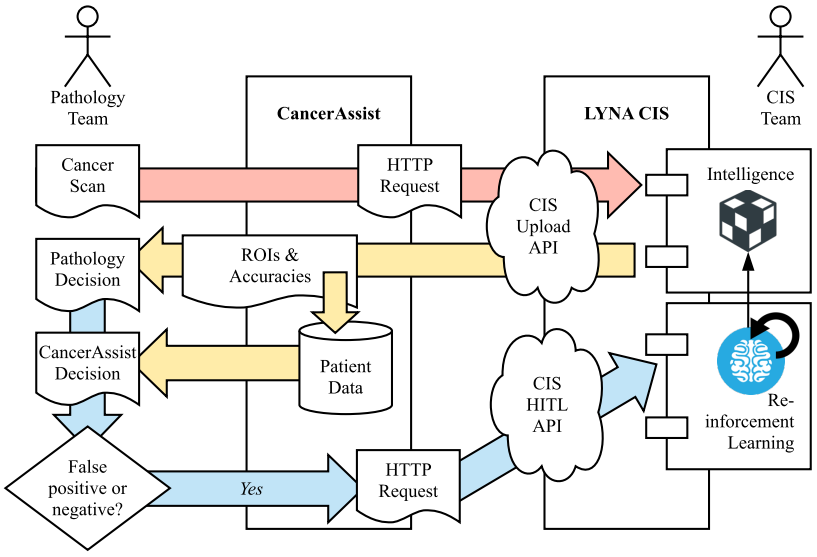


Figure 1.4: CancerAssist Context Diagram. ***Key:*** *Red Arrows = Scan Input; Yellow Arrows = Decision Output; Blue Arrows = HITL Feedback Input.*

Powering CancerAssist is Google AI's Lymph Node Assistant (LYNA) [140], a CNN based on the Inception-v3 model [125, 230]. To provide intelligence to CancerAssist, LYNA is hosted on a CIS, and thus the developers of CancerAssist call the relevant CIS API endpoints, in conjunction with extra information such as patient data and medical history, to produce the verdict. In the case of a positive verdict, the relevant ROIs CancerAssist has found are highlighted 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 CIS. As a result, they are unaware when updates to the model occur, nor do they know what training data they could provide to test their own 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 all the ontology tests that need to be assessed 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 CIS. 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 a CIS may come into mind:

- When is the model updated and how do the CIS team communicate that the model is updated?
- What benchmark test set of data do I use to ensure that the changed model doesn't affect other results?
- How do we know that the assumptions made by the CIS team who train model are correct?

Thus, improved documentation and additional metadata may be needed to better improve communication between developers and CIS providers. Such claims are further expanded upon in the following section.

1.2 Research Outcomes

In this work, we present a framework to improve the documentation quality of CVCISs and their APIs. We demonstrate that developers currently lack the understanding of how these services function and our framework mitigates this by presenting a solution to improve the documentation quality of the APIs and improve the existing techniques used to integrate these services into developer's end-applications.

The goals of this study aim to provide a snapshot of current developer practices towards the usage of CISs. This allows us to develop a guiding framework and recommendations for application developers and CISs providers alike. Our anchoring perspective is software quality—specifically, validation and verification—within such systems and what best practices within the field of software engineering can be applied to assist in consumption of CISs. Based on the motivating case studies in Section 1.1, we articulate three Research Hypotheses (RH1–3) below and seven Research Questions (RQs) based on both empirical and non-empirical software engineering methodology [215, 216].

RH1: ***Existing CISs present insufficient API documentation for general use.***

Research Hypothesis API documentation of intelligent services are inadequate and insufficient given the disparity of mindsets between the application developers and CIS providers. Chiefly, application developers have limited general understanding of the 'magic' that occurs behind these probabilistic 'intelligent' APIs. We do not know what key aspects of the documentation matter to them, nor what they do or do not understand of the existing documentation.

Research Goal To improve the effectiveness of the documentation in existing CIS providers, specifically of CVCIS APIs.

Research Questions

- **RQ1.1.** What practices are in use for intelligent services' API documentation?
- **RQ1.2.** How do developers currently understand and interpret the documentation given a lack of formal training in artificial intelligence? That is, what do they understand and not understand, and what key aspects of the API documentation matter do developers as they see it?
- **RQ1.3.** What additional information or attributes would developers prefer to be included in the API documentation?

Research Contribution An intelligent service API documentation quality assessment framework to evaluate how well the service has been documented for software engineers to use.

RH2: ***Existing CISs present insufficient metadata for context-specificity.***

Research Hypothesis Intelligent service APIs respond with insufficient information for developers to operationalise the service into a business-driven application and, thus, additional metadata is needed to assist developers. Such metadata is likely to be needed as part of the response objects of the API.

Research Goal To improve the quality of *context-specific response data* from the API endpoints of intelligent services.

Research Questions

- **RQ2.1.** What are current problems due to lack of return metadata?
- **RQ2.2.** What additional metadata do developers desire to achieve implementing context-specific applications?

►

RH2: ***Existing CISs present insufficient metadata for context-specificity.*** **(cont)**

Research Contributions A list of metadata key-value-pairs that assist developers in using these APIs during the development of software that consume these services. In essence, improvements to the framework of Research Outcome 1: "*An intelligent service API documentation* and metadata *quality assessment framework*".

RH3: ***RH1 and RH2 improve quality, productivity or developer informativeness.***

Research Hypothesis The implication of hypotheses 1 and 2 suggest that improving both the documentation and providing further metadata will improve product quality (internal or external), and/or developer productivity and/or developer education in developing software with intelligent components.

Research Goal To confirm if improvements to API documentation and response metadata are reflected as improvements to product quality, developer productivity and/or developer education.

Research Questions

RQ3.1. Does an improvement of documentation or metadata correlate to an improvement in software quality, developer productivity and/or developer informativeness?

RQ3.2. With respect to RQ3.1, the three aspects are explored:

(a) Does the improvement cause increased product quality, as measured through improved external quality metrics?

(b) Does the improvement cause increased developer productivity, as measured through improved internal quality metrics?

(c) Does the improvement cause increased developer informativeness or increased confidence in developing CIS-powered applications?

Research Contribution A concrete sample solution or framework that improves such metrics, thereby confirming that our documentation and metadata quality assessment framework improves these facets.

1.3 Concluding Remarks

Ultimately, we seek to understand the conceptual understanding of software engineers who operationalise stochastic and probabilistic systems, and furthermore

understand knowledge representation with these systems' API documentation. Our motivation is to provide insight into current practices and compare the best practices with actual practise. We strive for this to provide developers with a guiding framework on how to best operationalise these systems via the form of some checklist or tool they can use to ensure optimal software quality.

It is anticipated that the findings from this study in the CVCISs space will be generalisable to other areas, such as time-series information, natural language processing and others.

1.4 Contributions

1.5 Structure

CHAPTER 2

Background

In Chapter 1, we defined a common set of (artificial) intelligence-based cloud services that we label CISs. Specifically, we scope the primary body of this study's work on CVCISs (e.g., Google Cloud Vision [268], AWS Rekognition [270], Azure Computer Vision [269], Watson Visual Recognition [272] etc.). We claim developers have a distinctly deterministic mindset (2 + 2 *always* equals 4) whereas a CIS'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., [186, 221]) that apply to resolve this mismatch and thereby improve quality, such as VV? Chiefly, we anchor this question around three lenses of software engineering: creating a CIS, using a CIS, and the nature of CISs themselves.

Our chief concern lies with interaction and integration between CIS providers and consumers, the nature of applications built using a CIS, and the impact this has on software quality. We triangulate this around three pillars, which we diagrammatically represent in Figure 2.1.

1. **The development of the CIS.** We investigate the internal quality attributes of creating a CIS from the CIS *provider's* perspective. That is, we ask if existing verification techniques are sufficient enough to ensure that the CIS being developed actually satisfies the CIS 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 CIS.** We investigate the external quality attributes of using a CIS from the CIS *consumer's* perspective. That is, we ask if existing validation

techniques are sufficient enough to ensure that the end-users can actually use a CIS to build their software in the ways they expect the CIS to work.

3. **The nature of a CIS.** 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.

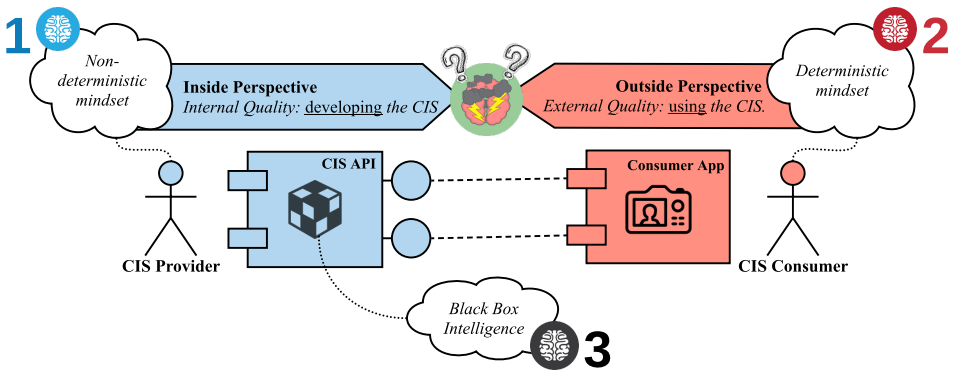


Figure 2.1: The three pillars by which we anchor the background: (1) developing a CIS with a non-deterministic mindset by the CIS provider; (2) the use of a CIS with a deterministic mindset by the CIS consumer; (3) the nature of a CIS itself.

Does a clash of deterministic consumer mindsets who use a CIS 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 core principles due to such mindset mismatch from the anchoring perspective of software quality, particularly around VV.

2.1 Software Quality

> *Quality... you know what it is, yet you don't know what it is.*
>
> Robert Pirsig, 1974 [185]

The philosophical viewpoint of 'quality' remains highly debated and there are multiple facets to perceive this complex concept [81]. Transcendentally, a viewpoint like that of Pirsig'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 International Organization for Standardization provides a breakdown of seven universally-applicable principles that defines quality for organisations, developers, customers and training providers [103]. More pertinently, the 1986 ISO standard for quality was simply "the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs" [102].

Using this sentence, what characteristics exist for non-deterministic CISs like that of a CVCIS? 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' [46, 83], 'meeting or exceeding customer expectation' [174], or 'fitness for use' [113]—but these then still depend on the solution description or requirements specification, and thus the same questions still apply.

Software quality is somewhat more concrete. Pressman [186] adapted the manufacturing-oriented view of quality from [19] 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 CIS, however, are any of the above actually guaranteed? Given that the core of a system built using a CIS 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 VV.

2.1.1 Validation and Verification

To explain VV, we analogously recount a tale given by Pham [183] on his works on reliability. A high-school student sat a standardised test that was sent to 350,0000 students [231]. A multiple-choice algebraic equation problem used a variable, a, and intended that students *assume* that the variable was non-negative. Without making this assumption explicit, there were two correct answers to the multiple choice answer. Up to 45,000 students had their scores retrospectively boosted by up to 30 points for those who 'incorrectly' answered, however, outcomes of a student's higher education were, thereby, affected by this one oversight in quality assessment. The examiners wrote a poor question due to poor process standards to check if their 'correct' answers were actually correct. The examiners "didn't build the right product" nor did they "build the product right" by writing an poor question and failing to ensure quality standards, in the phrases Boehm [22] coined.

This story describes the issues with the cost of quality [21] and the importance of VV: just as the poorly written exam question had such a high toll the 45,000

unlucky students, so does poorly written software in production. As summarised by Pressman [186], data sourced from Cigital Inc. [42] 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 [233].

Formally, we refer to the IEEE Standard Glossary of Software Engineering Terminology [100] for to define VV:

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 a CIS, we have two perspectives on VV: that of the API provider and consumer (Figure 2.2).

The verification process of API providers 'leak' out to the context of the developer's project dependent on the CIS. Poor verification in the *internal quality* of the CIS will entail poor process standards, such as poor definitions and terminology used, support tooling and description of documentations [221]. Though it is commonplace for providers to have a 'ship-first-fix-later' mentality of 'good-enough' software [240], 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 VV in the context of web services on the cloud [34, 35, 71, 92, 160, 162, 208, 255], though little have recently considered how adding 'intelligence' to these services affects existing proposed frameworks and solutions. For a CVCIS, what might this entail? Which assurances are given to the consumers, and how is that information communicated? To verify if the service is working correctly, does that mean that we need to deploy the system first to get a wider range of data, given the stochastic nature of the black box?

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

Therefore, just as the test answers with were both correct and incorrect at the same time, so is the same with CISs returning a probabilistic result: no result is certain. While VV has been investigated in the area of mathematical and earth sciences for numerical probabilistic models and natural systems [170, 207], from

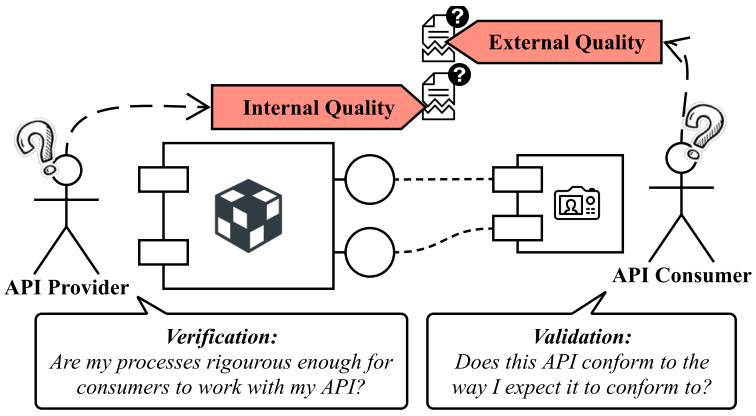


Figure 2.2: The 'leakage' of internal quality into the API consumer's product and external quality imposing on the API provider.

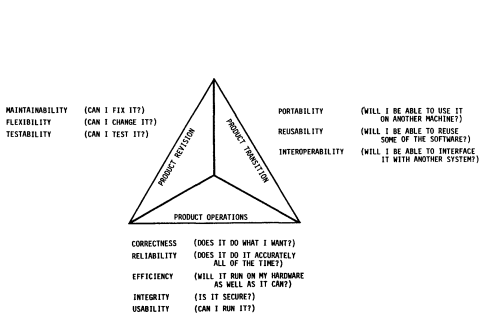
the software engineering literature, little work has been achieved to look at the surrounding area of probabilistic systems hidden behind API calls.

Now that a developer is using a probabilistic system behind a deterministic API call, what does it mean in the context of VV? 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, VV must be re-thought out extensively.

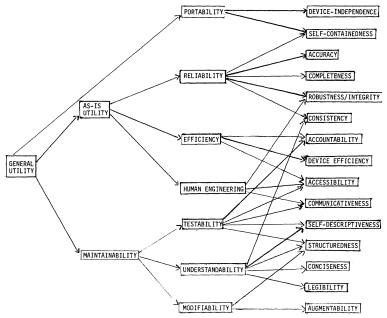
2.1.2 Quality Attributes and Models

Similarly, quality models are used to capture internal and external quality attributes via measurable metrics. Is a similar issue reflected from that of VV 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.

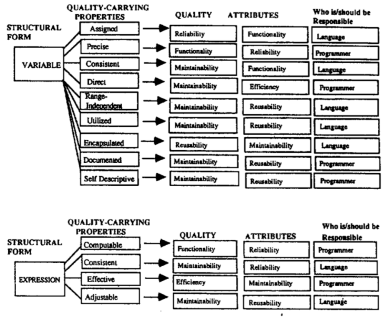
Quality attribute assessment models (like those shown in Figure 2.3) are an early concept in software engineering, and systematically evaluating software quality appears as early as 1968 [206]. Rubey and Hartwick's 1968 study introduced the phrase 'attributes' as a "prose expression of the particular quality of desired software" (as worded by Boehm et al. [23]) and 'metrics' as mathematical parameters on a scale of 0 to 100. Early attempts to categorise wider factors under a framework was proposed by McCall, Richards, and Walters in the late 1970s [38, 146]. 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?*)



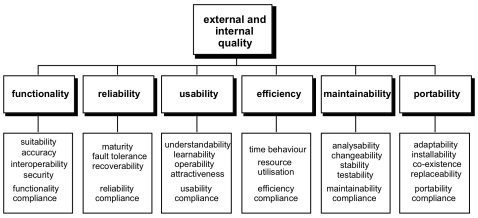
(a) McCall's quality software factors (1977) [146].



(b) Bohem's software quality characteristics tree (1978) [23].



(c) Dromey's quality-carrying properties and programming languages (1995) [62].



(d) ISO/IEC software product evaluation characteristics (1999) [104].

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

and operation (*how effective is the system at achieving its tasks?*) (Figure 2.3a). The model also introduced 11 attributes alongside numerous direct and indirect measures to help quantify quality. This model was further developed by Boehm et al. [23] who independently developed a similar model, starting with an initial set of 11 software characteristics. It further defined candidate measurements of Fortran code to such characteristics, taking shape in a tree-like structure as in Figure 2.3b. In the mid-1990s, Dromey's interpretation [62] defined a set of quality-carrying properties with structural forms associated to specific programming languages and conventions (Figure 2.3c). The model also supported quality defect identification and proposed an improved auditing method to automate defect detection for code editors in IDEs. As the need for quality models became prevalent, the International Organization for Standardization standardised software quality under ISO/IEC-9126 [104] (the Software Product Evaluation Characteristics, Figure 2.3d), which has since recently been revised to ISO/IEC-25010 with the introduction of the SQUARE model [101], 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 [4].

Of all the models described, there is one quality attribute that relates most with our narrative of CIS quality: reliability. The definition of reliability is similar among

all quality models:

McCall et al. Extent to which a program can be expected to perform its intended function with required precision [147].

Boehm et al. Code possesses the characteristic *reliability* to the extent that it can be expected to perform its intended functions satisfactorily [23].

Dromey Functionality implies reliability. The reliability of software is therefore dependent on the same properties as functionality, that is, the correctness properties of a program [62].

ISO/IEC-9126 The capability of the software product to maintain a specified level of performance when used under specified conditions [104].

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 a CIS 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 CIS reliability; a CVCIS 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?

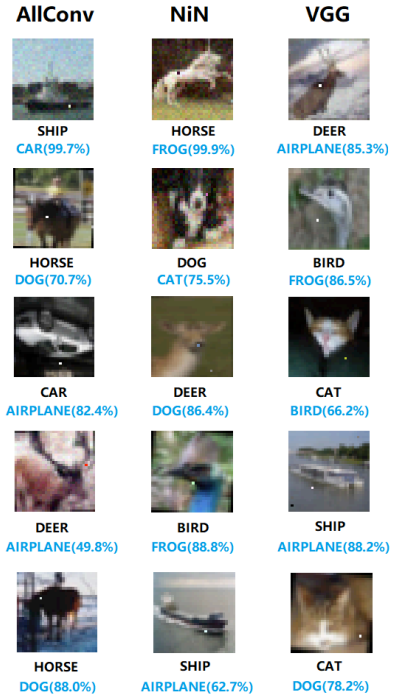
Moreover, defining these systems in themselves is challenging when requirements specifications and solution descriptions are dependent on nondeterministic and probabilistic algorithms. We discuss this further in Section 2.2.

2.1.3 Reliability in Computer Vision

Testing computer vision deep-learning reliability is an area explored typically through the use of adversarial examples [229]. These input examples are where images are slightly perturbed to maximise prediction error but are still interpretable to humans. Refer to Figure 2.4.

Google Cloud Vision, for instance, fails to correctly classify adversarial examples when noise is added to the original images [96]. Rosenfeld et al. [204] illustrated that inserting synthetic foreign objects to input images (e.g., a cartoon elephant) can alter classification output. Wang et al. [243] 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. Su et al. [223] 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, Eykholt et al. [66] recently showed similar results that

(a) Adding 10% impulse noise to an image of a teapot changes Google Cloud Vision's label from *teapot* (above) to *biology* (below) [96].



(b) One-pixel attacks applied to three NN: AllConv, NiN and VGG [223].

(c) Adversarial examples to trick face recognition from the source to target images [243].

Figure 2.4: Sample adversarial examples.

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 [67, 127], "the real world may be adversarial enough" [182] 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 a CIS are treated as though they are nondeterministic; Chapter 2 introduces CISs as essentially black-box behaviour that can change over time. As such, we adopt the nondeterministic behaviour that they present.

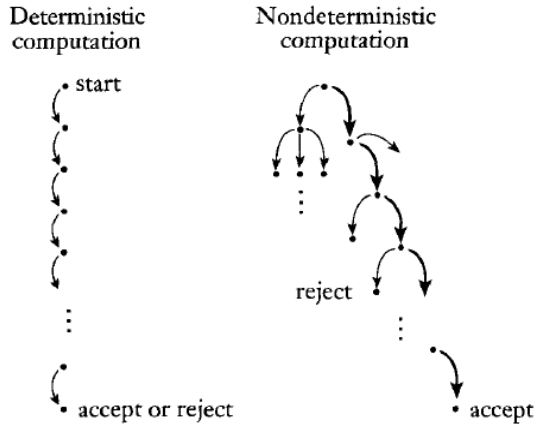


Figure 2.5: A deterministic system (left) always returns the same result in the same amount of steps. A nondeterministic system does not guarantee the same outcome, even with the same input data. Source: [70].

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 Quinlan [188] and Michie [153]), and although there has since been a significant body of work in the area [10, 11, 17, 24, 32, 51, 68, 77, 111, 135, 138, 145, 177, 191, 205, 218, 239, 241], it is evident that 'accuracy' or model 'confidence' is still used as a primary criterion for AI evaluation [97, 106, 220]. Much research into

NN or 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 from past data" [73], a problem commonly referred to as the *datashift* problem [226]. Analogously, Freitas [73] 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.*** [73]

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 [138]), what is known is (i) there exists a critical trade-off between accuracy and interpretability [57, 72, 87, 110, 116, 257], and (ii) a single quantifiable value cannot satisfy the subjective needs of end-users [73]. As ever-growing domains ML become widespread[1], 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 [36, 98, 117]), 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[2]) are not realised: model optimisation is only worthwhile if they "actually solve the original [human-centred] task of providing explanation" [161] to end-users. **Therefore, when human-decision makers must be interpretable themselves [194], any AI they depend on must also be interpretable.**

Recently, discussion behind such a notion to provide legal implications of interpretability is topical. Doshi-Velez et al. [60] discuss when explanations are not provided from a legal stance—for instance, those affected by algorithmic-based decisions have a 'right to explanation' [86, 242] under the European Union's GDPR[3]. But, explanations are not the only way to ensure AI accountability: theoretical

[1] In areas such as medicine [16, 32, 65, 107, 111, 132, 178, 193, 239, 254, 259], bioinformatics [56, 74, 108, 115, 228], finance [11, 54, 98] and customer analytics [135, 241].

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

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

guarantees (mathematical proofs) or statistical evidence can also serve as guarantees [60], however, in terms of explanations, what form they take and how they are proven correct are still open questions [138].

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' [148] into programs today has been achieved, but *decoding* that information is what still remains problematic. At a high level, Shannon and Weaver's theory of communication [213] applies, just as others have done with similar issues in the SE realm [156, 249] (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 [60] and the AI interprets the tree inversely to humans. Therefore, the interpretation or explanation is done inversely: humans do not explain the individual neurons fired to explain their predictions, and therefore the algorithmic transparent explanations of AI algorithms (*"which neurons were fired to make this AI think this tree is a tree?"*) do not work here.

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

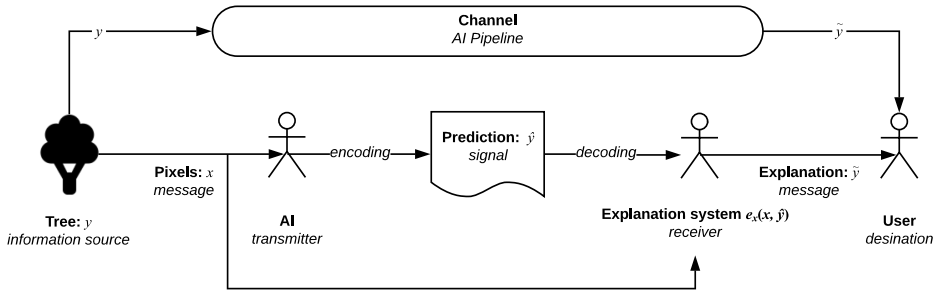


Figure 2.6: Theory of AI communication from information source, y, to intended user as explanations $\tilde{y}$.

2.2.3 Mechanics of Model Interpretation

How do we interpret models? Methods for developing interpretation models include: decision trees [29, 44, 90, 187, 201], decision tables [12, 135] and decision sets [129, 161]; input gradients, gradient vectors or sensitivity analysis [11, 133, 191, 205, 211]; exemplars [75, 118]; generalised additive models [36]; classification (*if-then*) rules [26, 43, 172, 236, 251] and falling rule lists [218]; nearest neighbours [145, 212, 227, 248, 258] and Naïve Bayes analysis [16, 40, 76, 93, 124, 132, 152, 259].

Cross-domain studies have assessed the interpretability of these techniques against end-users, measuring response time, accuracy in model response and user confidence [5, 74, 91, 98, 145, 209, 224, 241], although it is generally agreed that decision rules and decision tables provide the most interpretation in non-linear models such as SVMs or NNs [74, 145, 241]. For an extensive survey of the benefits and fallbacks of these techniques, we refer to Freitas [73], Doshi-Velez et al. [60] and Doshi-Velez and Kim [59].

2.3 Application Programming Interfaces

APIs are the interface between a developer needs and the software components at their disposal [7] 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 a CIS) is therefore directly related to the quality the API offers [123].

Good APIs are known to be intuitive and require less documentation browsing [184], 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 [94]) that, ultimately, causes the maintenance to be far more expensive, a phenomenon widely known in software engineering economics (see Section 2.1.1).

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 **??**, Appendix A.1.

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 machine learning techniques in computer vision cannot effectively formulate queries to help bridge those gaps in their understanding to figure out more about the CVCIS 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 [234]. 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 [27, 58, 159, 199] though these studies have not effectively communicated *why* these artefacts are important. What makes the developer *conceptually understand* these artefacts?

Robillard and Deline [199] 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 [198]. 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 APIâĂŹs implementation consumes resources, reports errors and has side effects.* [199]

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*" [167]. This thereby reiterates the conceptual understanding of an API as paramount.

A later study by Ko and Riche [122] 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 CIS 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. Ko and Riche 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 CIS 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 Chapter 3) in the search for further insight into the developer's approach toward existing CIS 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 [63]. 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 Section 1.2) 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 Easterbrook et al. [63], Wohlin and Aurum [252], Wohlin et al. [253] and Shaw [214].

3.1 Research Questions Revisited

In Section 1.2, we introduce three hypothesis of this study (RH1–RH3), namely: (i) existing CIS APIs are poorly documented for general use (RH1); (ii) existing CIS 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 Easterbrook et al. [63], a technique originally proposed in the field of psychology by Meltzoff [150] 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 [216]. 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 [250].

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 a CIS 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 RQ1.1 and RQ2.1 respectively describe and classify what practices are in use for existing CIS 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. RQ1.2 is a descriptive-process question to help us understand how the developer currently understands existing CIS 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, RQ3.1 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 RQ3.2, 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

RQ1.3 and RQ2.2 are both *design questions*; they are concerned with ways in which we can improve a CIS 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 a CIS.

3.2 Philosophical Stances

Philosophical stances guide the researcher's action by fortifying what constitutes 'valid truth' against a fundamental set of core beliefs [196]. In software engineering, four dominant philosophical stances are commonly characterised [45, 180]: 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 [121]. 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 [33]. 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" [63], and this study utilises such an approach to shift the mindset of CIS 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" [63]. Creswell and Creswell [45] suggest 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 [217]. The mixed-methods approach combines five classes of field study methods (or empirical strategies/studies) most relevant in empirical software engineering research [63, 114, 253]: 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 [109] 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 Section 3.1 as presented by Easterbrook et al. [63].

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 [256]. 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 (RQ2.1 and RQ2.2) in exploring whether the context of an application using a CIS requires the CIS 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 CIS APIs, we can use survey research to assist in answering our exploratory and base-rate research questions under RH1 and RH2 (see Section 3.1.1) 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. 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 [200]. 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 [50] 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 A2I2 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 CVCISs; 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

Singer et al. developed a taxonomy [134, 217] 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 Figure A.4.

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 (Sections 3.3.1 and 3.3.2), our approach to analysing the data, as well as linking the experiment back to a research hypothesis and question (Section 1.2). A high-level overview of the proposed experiments and the major bodies of work they encompass is presented in Figure 3.1.

3.4.1 Experiment I: Develop Initial Framework

Experiment I shapes a context-agnostic approach to understand current usage patterns of CIS 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.

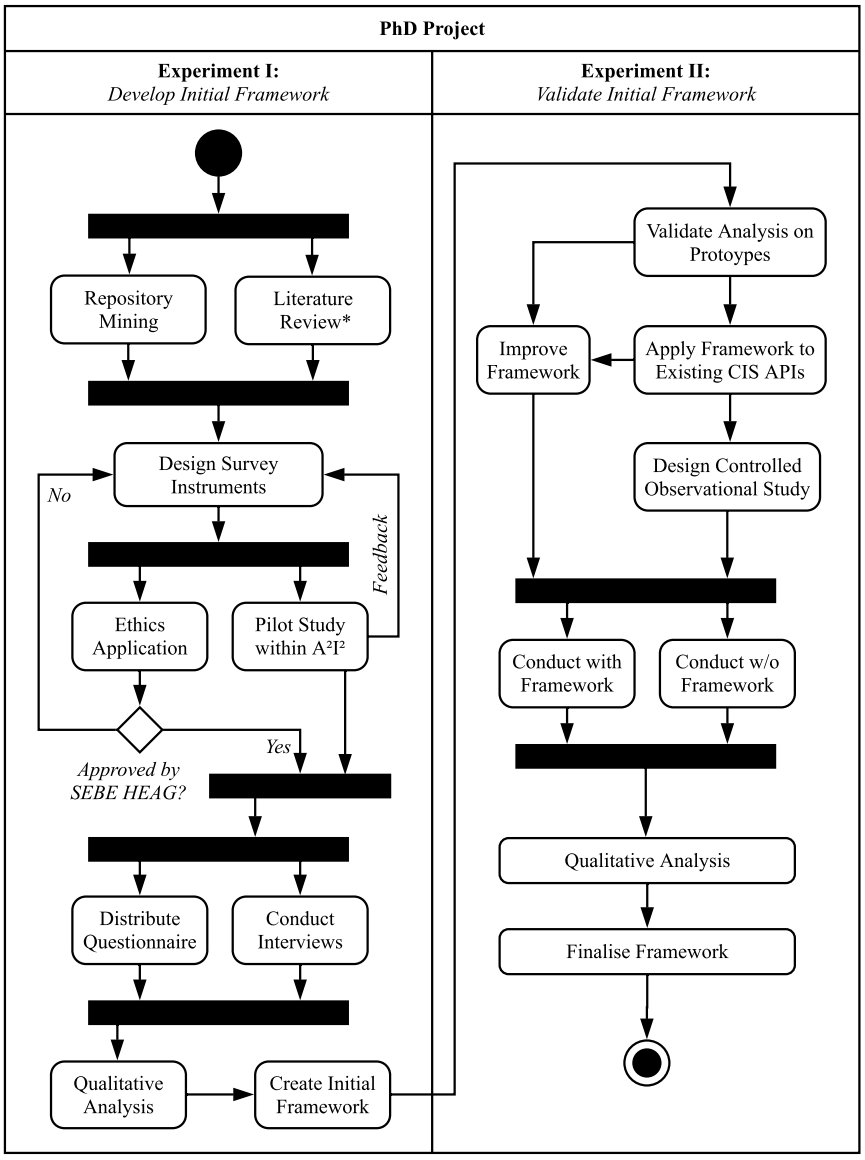


Figure 3.1: High-level activity diagram of the proposed experiments in this study. Literature review is ongoing.

Relevance and Motivation

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

RH1 postulates that the software engineering community do not fully understand the 'magic' behind CIS APIs. As described in Section 1.2, they face a gap in their

understanding around the underlying architecture of pre-built, machine learnt APIs (RQ1.2). Software developers are not well-supported by the CIS providers, and therefore do not have a consistent set of common best practices when approaching to use these APIs (RQ1.1). It is therefore necessary that CIS providers provide additional information to gap this mismatched understanding (RQ1.3).

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 and Quora, as well as writing their experiences in personal blogs such as Medium. The simplest of these platforms is Stack Overflow (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, Stack Overflow is often acknowledged as *the* 'go-to' place for developers to find high-quality code snippets that assist in their problems [225].

Thus, to begin validating CIS API usage and misunderstanding in a generalised context (i.e., context-agnostic to the project at hand), we propose using repository mining on Stack Overflow to help answer our research questions. Specifically, we select Stack Overflow 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 Stack Overflow to mine developer discourse [6, 13, 41, 137, 165, 173, 189, 202, 219, 232, 244**?**].

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 Singer et al. [217] and Miles et al. [154] on coding and analysing this qualitative data gathered.

Phase 2: Personal Opinion Surveys We follow the triangulation approach proposed by Jick [109] to corroborate the qualitative data of developers' discussion of Stack Overflow with secondary survey research, thereby validating what people say on Stack Overflow with what is said and done in real life. Kitchenham and Pfleeger [120] 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

[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 Stack Overflow'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.

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.

As is good practice in developing questionnaire instruments to evaluate their reliability and validity [139], we evaluate our instrument design by asking colleagues to critique it via pilot studies within A2I2. 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 A2I2.

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 A2I2. 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 A2I2 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., [131]), 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 CIS APIs of differing contexts; (ii) presenting a solution framework to developers to interpret the improvement of their understanding when using a CIS.

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 CIS 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 (RQ2.1) and what types of metadata developers seek (RQ2.2).

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 (RQ3.1 and RQ3.2).

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 CIS 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 [210] to capture firsthand behaviours and interactions toward how software engineers approach using a CIS with and without our framework applied. We will provide improved documentation and metadata responses of a set of popular CVCISs 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 [53], maps and networks [154] 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 Section 3.2, we state that this study primarily adopts a critical theorist stance. Critical theorists assess research quality by the utility of the knowledge gained [63]. Lau [131] 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 Creswell and Creswell [45] 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 [61, 197] and careful consideration of this effect must be reflected when designing our controlled observation (Section 3.4.3). 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 [217]. 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 Jick's concurrent triangulation method [109] 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 [112] 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, Singer et al. [217] 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% [14]. 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 A2I2, 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 [126] or using a summating rating scale [222] 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 [120]. Therefore, designing a well-defined sample frame to determine which developers we wish to target is empirical. For this, we refer to Kitchenham and Pfleeger [120].

Student Cohorts

External validity is typically undermined when students are recruited in software engineering research, which is common practice [63]. 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 A2I2—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. Easterbrook et al. [63] 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 CIS 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, are now available as intelligent services 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 intelligent services; in particular, their behavioural consistency and transparency of their functionality. We evaluated the responses of three different intelligent services (specifically computer vision) 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 intelligent service providers to inform risk and assist maintainability.

4.1 Introduction

The availability of intelligent services has made AI tooling accessible to software developers and promises a lower entry barrier for their utilisation. Consider state-of-the-art computer vision 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, intelligent services providing computer vision (e.g., [268–281]) abstract these complexities behind a web API call. This removes the need to understand the

[†]This chapter is originally based on A. Cummaudo, R. Vasa, J. Grundy, M. Abdelrazek, and A. Cain, "Losing Confidence in Quality: Unspoken Evolution of Computer Vision Services," in *35th IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, OH, USA, Sep. 2019. Terminology has been updated to fit this thesis.

complexities required of ML, and requires little more than the knowledge on how to use RESTful endpoints. The ubiquity of these services is exemplified through their rapid uptake in applications such as aiding the vision-impaired [49, 190].

While intelligent services 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—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 SE practices advocate maintainability and reliability as primary characteristics; stability, testability, fault tolerance, changeability and maturity are all concerns for quality in software components [95, 186, 221] and one must factor these in with consideration to software evolution challenges [52, 85, 151, 235, 237]. However, the effect this non-deterministic behaviour has on quality when masked behind an intelligent service is still under-explored to date in SE literature, to our knowledge. Where software depends on intelligent services 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 computer vision service may encapsulate deep-learning strategies or stochastic methods to perform image analysis, but developers are more likely to approach intelligent services with a mindset that anticipates consistency. Although the documentation does hint at this non-deterministic

[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 computer vision services.

behaviour (i.e., the descriptions of 'confidence' in various computer vision services suggest the they are not always confident, and thus not deterministic [282–284]), 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 computer vision services, as they currently stand, offer consistent behaviour, and if not, how is this conveyed to developers (if it is at all)? If computer vision services are to be used in production services, do they ensure quality under rigorous Software Quality Assurance (SQA) frameworks [95]? What evolution risk [52, 85, 151, 237] 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 intelligent services. We introduce a motivating example in Section 4.2, discussing related work and our methodology in Section 4.3 and **??**. We present and interpret our findings in Section 4.5. We argue with quantified evidence that these intelligent services can only be considered with a mature appreciation of risks, and we make a set of recommendations in Section 4.6.

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 computer vision engine from scratch, Rosa thinks she can achieve this using one of the popular computer vision services (e.g., [268–281]). However, Rosa comes from a typical software engineering background with limited knowledge of the underlying deep-learning techniques and implementations as currently used in computer vision. 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 computer vision services 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 computer vision services often present conceptually similar functionality but are behaviourally very different**. Intelligent service 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 computer vision services."** 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 computer vision services *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 computer vision services through the lenses of an SQA framework, robustness, consistency, and maintainability often feature as quality attributes in myriad software quality models (e.g., [104]). 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) [186]. 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 computer vision 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 [229], 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 [96]. Rosenfeld et al. [204] illustrated that inserting synthetic foreign objects to input images (e.g., a cartoon elephant) can completely alter classification output. Wang et al. [243] 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. Su et al. [223] 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, Eykholt et al. [66] recently showed similar results that made a CNN interpret a stop road-sign (with mimicked graffiti) as a 45mph speed limit sign.

The results suggest that current state-of-the-art computer vision 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 [67, 127], "the natural world may be adversarial enough" [182] to fool AI software. Though some limitations and guidelines have been explored in this area, the perspective of *intelligent* services is yet to be considered and specific guidelines do not yet exist when using computer vision services.

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" [8]. Murphy et al. [158] 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, Braiek and Khomh [25] 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., [28, 157, 164]) are targeted chiefly to the ML engineer, and not the application developer.

However, Arpteg et al. [8] 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., intelligent services) of the 'machine learning spectrum' [171], 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 Computer vision services 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 [78]: these attributes are exposed using Service Level Agreements (SLA) 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 intelligent services 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 intelligent service quality or assure quality of these computer vision services. Web services and their interfaces (APIs) are the bridge between developers' needs and the software components [7]; therefore, assessing such computer vision services from the quality of their APIs is thereby directly related to the development quality [123]. Good APIs should be intuitive and require less documentation browsing [184], 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 [245]. The consequences of addressing these concerns in development leads to a higher demand in technical support (as measured in [94]) that, ultimately, causes the maintenance to be far more expensive, a phenomenon widely known in software engineering economics [22]. 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 Ko and Riche, 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 Mitchell et al. [155] 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. Gebru et al. [82] 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 intelligent services, 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 computer vision services 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] Computer vision services do not respond with consistent outputs between services, given the same input image.

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

[RH3] Computer vision services 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 computer vision services: AWS Rekognition [270], Google Cloud Vision [268], Azure Computer Vision [269]. Specifically, we targeted the AWS `DetectLabels` endpoint [283], the Google Cloud Vision `annotate:images` endpoint [282] and Azure's `analyze` endpoint [284]. For the remainder of this paper, we de-identify our selected computer vision services 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 [195] and the consistent popularity of discussion amongst developers in developer communities such as Stack Overflow. While we choose these particular cloud computer vision services, we acknowledge that similar services [271–277] also exist, including other popular services used in Asia [278–281] (some offering 3D image analysis [285]). We reflect on the impacts this has to our study design in Section 4.7.

Our study involved an 11-month longitudinal study which consisted of two 13 week and 17 week experiments from April to August 2018 and November 2018 to March 2019, respectively. Our investigation into documentation occurred on August 28 2018. In total, we assessed the services with three data sets; we first ran a pilot study using a smaller pool of 30 images to confirm the end-points remain stable, re-running the study with a larger pool of images of 1,650 and 5,000 images. Our selection criteria for these three data sets were that the images had to have varying objects, taken in various scenes and various times. Images also needed to contain disparate objects. Our small data set was sourced by the first author by taking photos of random scenes in an afternoon, whilst our second data set was sourced from various members of our research group from their personal photo libraries. We also 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 [136]. We have made our other two data sets available online ([286]). We collected results and their responses from each service's API endpoint using a python script [287] that sent requests to each service periodically via cron jobs. Table 4.1 summarises various characteristics about the data sets used in these experiments.

We then performed quantitative analyses on each response's labels, ensuring all

Table 4.1: Characteristics of our data sets 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

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

We measure response consistency under 6 aspects:

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

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

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 [288].

4.5 Findings

4.5.1 Consistency of top labels

Consistency across services Table 4.2 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 Section 4.7), service vendors are not ‘plug-and-play’. If Rosa uploaded the sample images in this paper to her application to all services, she would find that only Figure 4.1 responds with ‘person’ for services B and C in their respective set of top labels. However, if she decides to then adopt service A, then Figure 4.1’s top label becomes ‘conversation’; the ‘person’ label does not appear within the top 15 labels for service A and, conversely, the ‘conversation’ label does not appear in the other services top

(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' } }.

15.

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

Hence, developers should ensure that the vocabulary used by a particular service is right for them before implementation. As each service does not work to the same standardised model, trained with disparate training data, and tuned differently, results will differ despite the same input. This is unlike deterministic systems: for example, switching from AWS Object Storage to Google Cloud Object storage will conceptually provide the same output (storing files) for the same input (uploading files). However, computer vision services 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 Figure 4.2a; '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 Figure 4.3a, service C presents a reasonably high confidence for the set of 7 top labels it returns,

(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.

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 Figure 4.3b, 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."* [289]

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 Figure 4.3b) may be missed.

4.5.2 Consistency of confidence

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

Therefore, Rosa could generally expect that the top labels she receives in her images do have high confidence. That is, each service will return a top label that

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]

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 Figure 4.6, the three services agree with $\sigma = 0.277$, significantly larger than that of all images in general $\sigma = 0.0831$. Figure 4.5 displays the cumulative distribution of all intersecting top labels' confidence values, presenting slightly similar results to that of Figure 4.4.

4.5.3 Evolution risk

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. Table 4.3 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 intelligent service 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 intelligent service 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

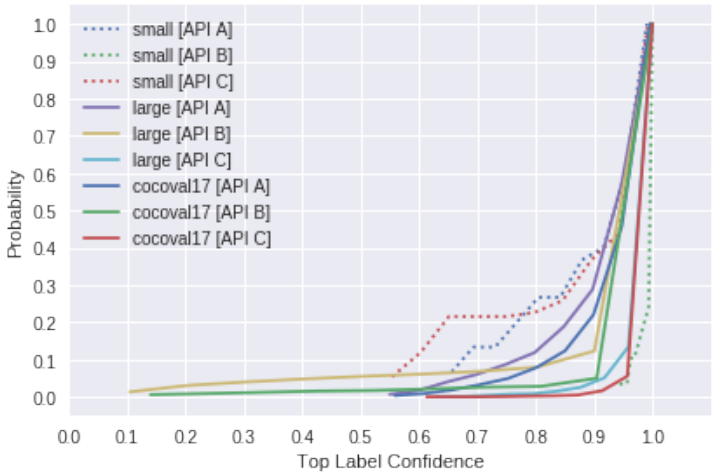


Figure 4.4: Cumulative distribution of the top labels' confidences. One in nine images return a top label(s) confident to $\gtrapprox$ 97%, though there is a wider distribution for service A.

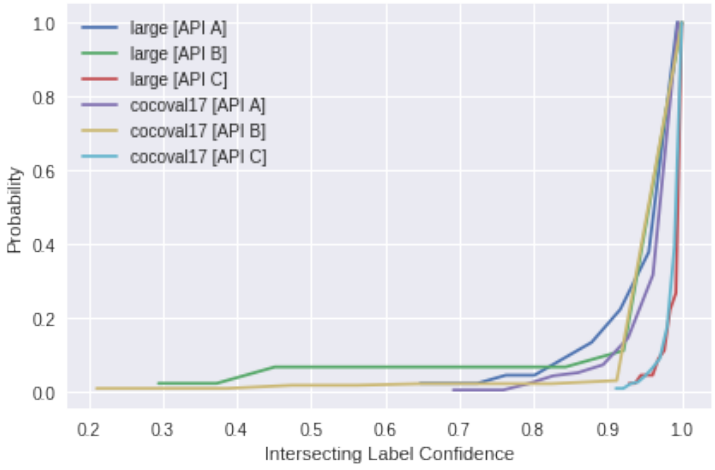


Figure 4.5: Cumulative distribution of intersecting labels top labels' confidences. The small data set is intentionally removed due to low intersections of labels (see Table 4.2).

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

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.

Intelligent services 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 Section 4.5.1 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 computer vision services. Developers should avoid their use in safety critical systems due to the lack of visible changes.

4.6.2 Recommendations for intelligent service providers

Improve the documentation Rosa does not know that service A returns back 'carrot' for its top response, with service C returning 'spoon' (Figure 4.2b). 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 Figure 4.3a: 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 [82, 155] 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 intelligent service, 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 intelligent service 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 computer vision techniques may have limits in what they can and cannot detect, but this limitation can be exposed in the documentation to the developers.

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 computer vision services were assessed. As suggested in Section 4.4, we note that there are other computer vision services 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 Section 4.4) 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 [290].

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 Section 4.6 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 computer vision services 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 intelligent service vendors and developers.

Standardised software quality models (e.g., [104]) target maintainability and reliability as primary characteristics. Quality software is stable, testable, fault

tolerant, easy to change and mature. These computer vision services are, however, in a nascent stage, difficult to evaluate, and currently are not easily interchangeable. Effectively, the intelligent service 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 computer vision), 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 [168].

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 [151, 235]. 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 intelligent service 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 intelligent services as a consequence.

CHAPTER 5

Systematic Mapping Study of API Documentation Knowledge[†]

Abstract Good 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 API documentation is highly valuable to the software development process; good documentation facilitates productivity and thus quality is better engineered into the system [149]. 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*

[†]This chapter is originally based on A. Cummaudo, R. Vasa, and J. Grundy, "What should I document? A preliminary systematic mapping study into API documentation knowledge," in *13th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Porto de Galinhas, Brazil, Sep. 2019. Terminology has been updated to fit this thesis.

misunderstand"). In the latter case, what causes this confusion 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., [142, 166, 246]); there still lacks a consolidated effort to capture the knowledge and artefacts best suited to *manually write* API documentation. This paper presents outcomes from a preliminary work to address this gap and offers two 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.

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.

This paper is structured as thus: Section 5.2 presents related work in the area; Section 5.3 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; Section 5.4 present our primary studies and our proposed taxonomy; Section 5.5 describes the threats to validity of this work and Section 5.6 provides concluding remarks and the future directions of this study.

5.2 Related Work

Systematic mapping studies have previously been explored in the area of API usability and developer experience. Nybom et al. [166] recently performed a systematic mapping study 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, Nybom et al. produce a systematic mapping study on the types of *tooling* that exists to assist in producing and validating API documen-

tation. While this is a systematic study with key insights into the types of tooling produced, there is still a gap for a systematic mapping study in what *guidelines* have been produced by the literature in developing natural-language documentation itself, which our work has addressed.

Watson [246] 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 SE.

A taxonomy of knowledge patterns within API reference documentation by Maalej and Robillard [142] 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.

5.3 Method

Our taxonomy development consisted of two phases. Firstly, we conducted a SMS to identify and analyse API documentation studies, following the guidelines of Kitchenham and Charters [119] and Petersen et al. [181]. Following this, we followed the software engineering (SE) taxonomy development method devised by Usman et al. [238] 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 Section 5.4. RQ2 addresses methodologies by which these studies come to these conclusions to identify gaps in literature where future studies can potentially focus.

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

Automatic Filtering Informed by similar previous studies in SE [80, 84, 238], we begin by defining the SWEBOK [100] 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:

("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 [30, 119]. A total 4,501 results[4] were found, with 549 being duplicates. Table 5.1 displays our results in further detail (duplicates not omitted).

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);

[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`

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;

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 at `http://bit.ly/2MtsIuE`.**

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 Wieringa and Heerkens's taxonomy

Intra-rater reliability of our 133 shortlisted papers was tested using the test-retest approach [119] 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 [130].

Data Extraction Of the 21 primary studies, we conducted abstract key-wording adhering to Petersen et al.'s guidelines [181] 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 Wieringa and Heerkens [250]: 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 Singer et al.'s framework of SE field study techniques [217]. Singer et al. 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.

Table 5.2 describes our data extraction form, which was used to collect relevant data from each paper. Figure 5.1 maps each study to one (or more, if applicable) of methodologies plotted against Wieringa and Heerkens's research approaches.

5.3.2 Development of the Taxonomy

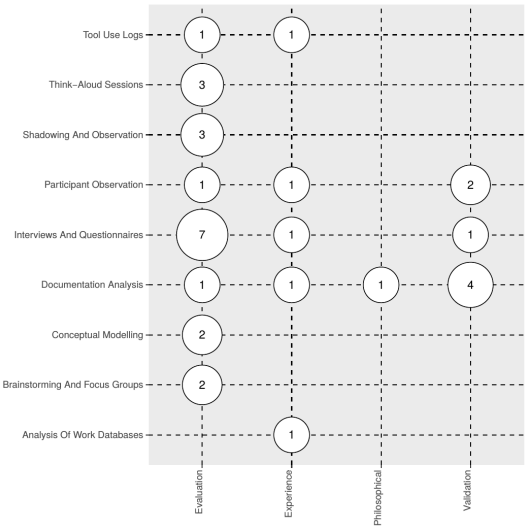


Figure 5.1: Systematic map: field study technique vs research type

Usman et al. concludes that a majority of SE taxonomies are developed in an ad-hoc way [238], 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 Usman et al.'s technique.

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 Section 5.3.1).

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 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 In the final phase of taxonomy development, taxonomy designers must **(13)** *validate the taxonomy* to assess its usefulness. Ideally, this is done by applying the taxonomy heuristically against developers or real-world case-studies. This remains a plan for future work (see Section 5.5).

5.4 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 Table 5.3, 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 Table 5.3 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 Table 5.3) 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].

5.5 Threats to Validity

Threats to *internal validity* concern factors internal to our study that may affect results. Guidelines on producing systematic reviews [119] 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. MacDonell et al. [143] 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 Garousi and Felderer [79]. 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 Section 5.6) 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 Section 5.3.1). 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.

5.6 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. Figure 5.1 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 Section 5.5) with an inter-rater reliability method;
- the development and applicability of our taxonomy will be further explored by triangulating the taxonomy against actual developers in industry to assess the efficacy of these recommendations—this will empirically reflect what is important from a *practitioner* point of view;
- reviewing the techniques and evaluation of our selected studies to extract the effectiveness of the various approaches used in the conclusions;
- conducting a heuristic validation of the taxonomy against intelligent APIs, given the current trend in SE that is exploring how machine learning and artificial intelligence-based applications may affect existing approaches;
- arrive at a relevance ranking for each of the 34 categories, based on developer surveys and current weights.

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: Dimensions A=Usage Description; B=Design Rationale; C=Domain Concepts; D=Support Artefacts; E=Documentation Presentation	Primary Studies	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%)
[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	[S2, S6, S13,	4/21 (19%)

Part III

Postface

CHAPTER 6

Conclusion

References

[1] "SOAP, Representational state transfer - Explore - Google Trends."

[2] *Pivotal Cloud Foundry, Google ML, and Spring*, Dec. 2017.

[3] *Machine learning with Google APIs*, Jan. 2019.

[4] R. E. Al-Qutaish, "Quality models in software engineering literature: an analytical and comparative study," *Journal of American Science*, vol. 6, no. 3, pp. 166–175, 2010.

[5] H. Allahyari and N. Lavesson, "User-oriented assessment of classification model understandability," in *11th scandinavian conference on Artificial intelligence*. IOS Press, 2011.

[6] M. Allamanis and C. Sutton, "Why, when, and what: analyzing stack overflow questions by topic, type, and code," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 53–56.

[7] K. Arnold, "Programmers are people, too," *ACM Queue*, vol. 3, no. 5, pp. 54–59, 2005.

[8] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software Engineering Challenges of Deep Learning," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, Oct. 2018, pp. 50–59.

[9] W. R. Ashby and J. R. Pierce, "An Introduction to Cybernetics," *Physics Today*, vol. 10, no. 7, pp. 34–36, Jul. 1957.

[10] M. G. Augasta and T. Kathirvalavakumar, "Reverse engineering the neural networks for rule extraction in classification problems," *Neural processing letters*, vol. 35, no. 2, pp. 131–150, 2012.

[11] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. MÃžller, "How to explain individual classification decisions," *Journal of Machine Learning Research*, vol. 11, no. Jun, pp. 1803–1831, 2010.

[12] B. Baesens, C. Mues, M. De Backer, J. Vanthienen, and R. Setiono, "Building Intelligent Credit Scoring Systems Using Decision Tables." *ICEIS*, 2003.

[13] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 112–121.

[14] Y. Baruch, "Response rate in academic studies—A comparative analysis," *Human relations*, vol. 52, no. 4, pp. 421–438, 1999.

[15] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[16] R. Bellazzi and B. Zupan, "Predictive data mining in clinical medicine: current issues and guidelines," *International journal of medical informatics*, vol. 77, no. 2, pp. 81–97, 2008.

[17] A. Ben-David, "Monotonicity maintenance in information-theoretic machine learning algorithms," *Machine Learning*, vol. 19, no. 1, pp. 29–43, 1995.

[18] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifier (URI): Generic syntax," Tech. Rep., 2004.

[19] J. Bessin, "The Business Value of Quality," *IBM developerWorks, June*, vol. 15, 2004.

[20] J. J. Blake, L. P. Maguire, B. Roche, T. M. McGinnity, and L. J. McDaid, "The Implementation of Fuzzy Systems, Neural Networks and Fuzzy Neural Networks using FPGAs." *Inf. Sci.*, 1998.

[21] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Foundations of empirical software engineering: the legacy of Victor R. Basili*, vol. 426, no. 37, 2005.

[22] B. W. Boehm, *Software engineering economics*. Prentice-hall Englewood Cliffs (NJ), 1981, vol. 197.

[23] B. W. Boehm, J. R. Brown, and H. Kaspar, "Characteristics of software quality," 1978.

[24] O. Boz, "Extracting decision trees from trained neural networks," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 456–461.

[25] H. B. Braiek and F. Khomh, "On testing machine learning programs," *arXiv preprint arXiv:1812.02257*, 2018.

[26] M. Bramer, *Principles of data mining*. Springer, 2007, vol. 180.

[27] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two studies of opportunistic programming: interleaving web foraging, learning, and writing code," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 1589–1598.

[28] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "WhatâĂŹs your ML Test Score? A rubric for ML production systems," 2016.

[29] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. CRC press, 1984.

[30] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, no. 4, pp. 571–583, Apr. 2007.

[31] M. Bunge, "A General Black Box Theory," *Philosophy of Science*, vol. 30, no. 4, pp. 346–358, Oct. 1963.

[32] A. Bussone, S. Stumpf, and D. O'Sullivan, "The Role of Explanations on Trust and Reliance in Clinical Decision Support Systems." *ICHI*, 2015.

[33] C. Calhoun, *Critical social theory: Culture, history, and the challenge of difference*. Wiley-Blackwell, 1995.

[34] G. Canfora, "User-side testing of web services," in *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on*. IEEE, 2005, p. 301.

[35] G. Canfora and M. Di Penta, "Testing services and service-centric systems: Challenges and opportunities," *It Professional*, vol. 8, no. 2, pp. 10–17, 2006.

[36] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible Models for HealthCare - Predicting Pneumonia Risk and Hospital 30-day Readmission." *KDD*, pp. 1721–1730, 2015.

[37] F. Casati, H. Kuno, G. Alonso, and V. Machiraju, "Web Services-Concepts, Architectures and Applications," 2003.

[38] J. P. Cavano, J. A. McCall, J. P. Cavano, J. A. McCall, J. P. Cavano, and J. A. McCall, "A framework for the measurement of software quality," *ACM SIGSOFT Software Engineering Notes*, vol. 3, no. 5, pp. 133–139, Nov. 1978.

[39] C. V. Chambers, D. J. Balaban, B. L. Carlson, and D. M. Grasberger, "The effect of microcomputer-generated reminders on influenza vaccination rates in a university-based family practice center," *The Journal of the American Board of Family Practice*, vol. 4, no. 1, pp. 19–26, 1991.

[40] J. Cheng and R. Greiner, "Learning bayesian belief network classifiers: Algorithms and system," in *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer, 2001, pp. 141–151.

[41] E. Choi, N. Yoshida, R. G. Kula, and K. Inoue, "What do practitioners ask about code clone? a preliminary investigation of stack overflow." in *IWSC*, 2015, pp. 49–50.

[42] Cigital Inc., "Case study: Finding defects earlier yields enormous savings," 2003.

[43] P. Clark and R. Boswell, "Rule induction with CN2: Some recent improvements," in *European Working Session on Learning*. Springer, 1991, pp. 151–163.

[44] M. Craven and J. W. Shavlik, "Extracting Tree-Structured Representations of Trained Networks." *NIPS*, 1995.

[45] J. W. Creswell and J. D. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017.

[46] P. B. Crosby, "Quality is free: The art of making quality free," *New York*, 1979.

[47] A. Cummaudo, R. Vasa, and J. Grundy, "What should I document? A preliminary systematic mapping study into API documentation knowledge," in *13th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Porto de Galinhas, Brazil, Sep. 2019.

[48] A. Cummaudo, R. Vasa, J. Grundy, M. Abdelrazek, and A. Cain, "Losing Confidence in Quality: Unspoken Evolution of Computer Vision Services," in *35th IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, OH, USA, Sep. 2019.

[49] H. da Mota Silveira and L. C. Martini, "How the New Approaches on Cloud Computer Vision can Contribute to Growth of Assistive Technologies to Visually Impaired in the Following Years," *Journal of Information Systems Engineering and Management*, vol. 2, no. 2, pp. 1–3, 2017.

[50] R. Davison, M. G. Martinsons, and N. Kock, "Principles of canonical action research," *Information systems journal*, vol. 14, no. 1, pp. 65–86, 2004.

[51] K. Dejaeger, F. Goethals, A. Giangreco, L. Mola, and B. Baesens, "Gaining insight into student satisfaction using comprehensible data mining techniques," *European Journal of Operational Research*, vol. 218, no. 2, pp. 548–562, 2012.

[52] S. Demeyer and T. Mens, *Software Evolution*. Springer, 2008.

[53] I. Dey, *Qualitative data analysis: A user friendly guide for social scientists*. Routledge, 2003.

[54] V. Dhar, D. Chou, and F. Provost, "Discovering Interesting Patterns for Investment Decision Making with GLOWER—A Genetic Learner Overlaid with Entropy Reduction," *Data Mining and Knowledge Discovery*, vol. 4, no. 4, pp. 251–280, 2000.

[55] V. C. Dibia, M. Ashoori, A. Cox, and J. D. Weisz, "TJBot," in *the 2017 CHI Conference Extended Abstracts*. New York, New York, USA: ACM Press, 2017, pp. 381–384.

[56] M. Doderer, K. Yoon, J. Salinas, and S. Kwek, "Protein subcellular localization prediction using a hybrid of similarity search and error-correcting output code techniques that produces interpretable results," *In silico biology*, vol. 6, no. 5, pp. 419–433, 2006.

[57] P. Domingos, "Occam's two razors: The sharp and the blunt," in *KDD*, 1998, pp. 37–43.

[58] B. Dorn and M. Guzdial, "Learning on the job: characterizing the programming knowledge and learning strategies of web designers," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 703–712.

[59] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017.

[60] F. Doshi-Velez, M. Kortz, R. Budish, C. Bavitz, S. Gershman, D. O'Brien, S. Schieber, J. Waldo, D. Weinberger, and A. Wood, "Accountability of AI Under the Law: The Role of Explanation," *arXiv.org*, Nov. 2017.

[61] S. W. Draper. The Hawthorne, Pygmalion, Placebo and other effects of expectation: some notes.

[62] R. G. Dromey, "A model for software product quality," *International Software Engineering Research Network*, vol. 21, no. 2, pp. 146–162, 1995.

[63] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to Advanced Empirical Software Engineering*. Springer Science & Business Media, Nov. 2007, pp. 285–311.

[64] B. Ehteshami Bejnordi, M. Veta, P. Johannes van Diest, B. van Ginneken, N. Karssemeijer, G. Litjens, J. A. W. M. van der Laak, and the CAMELYON16 Consortium, M. Hermsen, Q. F. Manson, M. Balkenhol, O. Geessink, N. Stathonikos, M. C. van Dijk, P. Bult, F. Beca, A. H. Beck, D. Wang, A. Khosla, R. Gargeya, H. Irshad, A. Zhong, Q. Dou, Q. Li, H. Chen, H.-J. Lin, P.-A. Heng, C. Haß, E. Bruni, Q. Wong, U. Halici, M. Ü. Öner, R. Cetin-Atalay, M. Berseth, V. Khvatkov, A. Vylegzhanin, O. Kraus, M. Shaban, N. Rajpoot, R. Awan, K. Sirinukunwattana, T. Qaiser, Y.-W. Tsang, D. Tellez, J. Annuscheit, P. Hufnagl, M. Valkonen, K. Kartasalo, L. Latonen, P. Ruusuvuori, K. Liimatainen, S. Albarqouni, B. Mungal, A. George, S. Demirci, N. Navab, S. Watanabe, S. Seno, Y. Takenaka, H. Matsuda, H. Ahmady Phoulady, V. Kovalev, A. Kalinovsky, V. Liauchuk, G. Bueno, M. M. Fernandez-Carrobles, I. Serrano, O. Deniz,

D. Racoceanu, and R. Venâncio, “Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer,” *JAMA*, vol. 318, no. 22, pp. 2199–12, Dec. 2017.

[65] W. Elazmeh, W. Matwin, D. O’Sullivan, W. Michalowski, and W. Farion, “Insights from predicting pediatric asthma exacerbations from retrospective clinical data,” in *Evaluation Methods for Machine Learning II–Papers from 2007 AAAI Workshop*, 2007, pp. 10–15.

[66] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, “Robust Physical-World Attacks on Deep Learning Visual Classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[67] ——, “Robust Physical-World Attacks on Deep Learning Visual Classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[68] A. J. Feelders, “Prior knowledge in economic applications of data mining,” in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2000, pp. 395–400.

[69] R. T. Fielding, “*Architectural Styles and the Design of Network-based Software Architectures* ,” Ph.D. dissertation, University of California, Irvine.

[70] I. Finalyson, “Nondeterministic Finite Automata,” 2018.

[71] H. Foster, S. Uchitel, J. Magee, and J. Kramer, “Model-based verification of web service compositions,” in *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*. IEEE, 2003, pp. 152–161.

[72] A. A. Freitas, “A critical review of multi-objective optimization in data mining: a position paper,” *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 2, pp. 77–86, 2004.

[73] ——, “Comprehensible classification models,” *ACM SIGKDD Explorations Newsletter*, vol. 15, no. 1, pp. 1–10, Mar. 2014.

[74] A. A. Freitas, D. C. Wieser, and R. Apweiler, “On the importance of comprehensible classification models for protein function prediction,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 7, no. 1, pp. 172–182, 2010.

[75] B. J. Frey and D. Dueck, “Clustering by Passing Messages Between Data Points,” *Science*, vol. 315, no. 5814, pp. 972–976, Feb. 2007.

[76] N. Friedman, D. Geiger, and M. Goldszmidt, “Bayesian network classifiers,” *Machine Learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[77] G. Fung, S. Sandilya, and R. B. Rao, “Rule extraction from linear support vector machines,” in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 32–40.

[78] S. K. Garg, S. Versteeg, and R. Buyya, “SMICloud: A Framework for Comparing and Ranking Cloud Services,” in *2011 IEEE 4th International Conference on Utility and Cloud Computing (UCC 2011)*. IEEE, Nov. 2011, pp. 210–218.

[79] V. Garousi and M. Felderer, “Experience-based guidelines for effective and efficient data extraction in systematic reviews in software engineering,” in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE’17. New York, NY, USA: ACM, 2017, pp. 170–179.

[80] V. Garousi, M. Felderer, and M. V. MÃđntylÃđ, “Guidelines for including grey literature and conducting multivocal literature reviews in software engineering,” *Information and Software Technology*, vol. 106, pp. 101 – 121, 2019.

[81] D. A. Garvin and W. D. P. Quality, “Really Mean,” *Sloan management review*, vol. 25, 1984.

[82] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumeé III, and K. Crawford, “Datasheets for datasets,” *arXiv preprint arXiv:1803.09010*, pp. 1–17, 2018.

[83] H. L. Gilmore, “Product conformance cost,” *Quality progress*, vol. 7, no. 5, pp. 16–19, 1974.

[84] R. L. Glass, I. Vessey, and V. Ramesh, “Research in software engineering: an analysis of the literature,” *Information and Software Technology*, vol. 44, no. 8, pp. 491–506, 2002.

[85] M. W. Godfrey and D. M. German, “The past, present, and future of software evolution,” in *2008 Frontiers of Software Maintenance*, Sep. 2008, pp. 129–138.

[86] B. Goodman and S. R. Flaxman, “EU regulations on algorithmic decision-making and a "right to explanation".” *IEEE Transactions on Evolutionary Computation*, 2016.

[87] P. D. Grünwald, *The minimum description length principle*. MIT press, 2007.

[88] M. J. Hadley, “Web application description language (WADL),” 2006.

[89] H. A. Haenssle, C. Fink, R. Schneiderbauer, F. Toberer, T. Buhl, A. Blum, A. Kalloo, A. B. H. Hassen, L. Thomas, A. Enk, L. Uhlmann, Reader study level-I and level-II Groups, C. Alt, M. Arenbergerova, R. Bakos, A. Baltzer, I. Bertlich, A. Blum, T. Bokor-Billmann, J. Bowling, N. Braghiroli, R. Braun, K. Buder-Bakhaya, T. Buhl, H. Cabo, L. Cabrijan, N. Cevic, A. Classen, D. Deltgen, C. Fink, I. Georgieva, L.-E. Hakim-Meibodi, S. Hanner, F. Hartmann, J. Hartmann, G. Haus, E. Hoxha, R. Karls, H. Koga, J. Kreusch, A. Lallas, P. Majenka, A. Marghoob, C. Massone, L. Mekokishvili, D. Mestel, V. Meyer, A. Neuberger, K. Nielsen, M. Oliviero, R. Pampena, J. Paoli, E. Pawlik, B. Rao, A. Rendon, T. Russo, A. Sadek, K. Samhaber, R. Schneiderbauer, A. Schweizer, F. Toberer, L. Trennheuser, L. Vlahova, A. Wald, J. Winkler, P. Wölbing, and I. Zalaudek, "Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists," *Annals of Oncology*, vol. 29, no. 8, pp. 1836–1842, May 2018.

[90] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*, ser. Data Mining, Inference, and Prediction. Springer Science & Business Media, Jan. 2001.

[91] B. Hayete and J. R. Bienkowska, "GOTrees - Predicting GO Associations from Protein Domain Composition Using Decision Trees." *Pacific Symposium on Biocomputing*, pp. 127–138, 2005.

[92] R. Heckel and M. Lohmann, "Towards contract-based testing of web services," *Electronic Notes in Theoretical Computer Science*, vol. 116, pp. 145–156, 2005.

[93] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie, "Dependency networks for inference, collaborative filtering, and data visualization," *Journal of Machine Learning Research*, vol. 1, no. Oct, pp. 49–75, 2000.

[94] M. Henning, "API design matters," *Commun. ACM*, vol. 52, no. 5, pp. 46–56, 2009.

[95] J. W. Horch, *Practical guide to software quality management*. Artech House, 2003.

[96] H. Hosseini, B. Xiao, and R. Poovendran, "Google's Cloud Vision API is Not Robust to Noise," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, Jan. 2018, pp. 101–105.

[97] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[98] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, "An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models," *Decision Support Systems*, vol. 51, no. 1, pp. 141–154, Apr. 2011.

[99] K. Hwang, *Cloud Computing for Machine Learning and Cognitive Applications: A Machine Learning Approach*. MIT Press, 2017.

[100] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," 1990.

[101] International Organization for Standardization, "Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models," 2011.

[102] ——, "Quality – Vocabulary," ISO/IEC, 1986.

[103] ——, "Quality management systems – Fundamentals and vocabulary," ISO/IEC, 2015.

[104] ——, "**Information technology – Software product quality**," Nov. 1999.

[105] A. Iyengar, "Supporting Data Analytics Applications Which Utilize Cognitive Services," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, May 2017, pp. 1856–1864.

[106] N. Japkowicz and M. Shah, *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.

[107] M. W. M. Jaspers, M. Smeulers, H. Vermeulen, and L. W. Peute, "Effects of clinical decision-support systems on practitioner performance and patient outcomes: a synthesis of high-quality systematic review findings," *Journal of the American Medical Informatics Association*, vol. 18, no. 3, pp. 327–334, 2011.

[108] T. Jiang and A. E. Keating, "AVID: an integrative framework for discovering functional relationships among proteins," *BMC bioinformatics*, vol. 6, no. 1, p. 136, 2005.

[109] T. Jick, *Mixing qualitative and quantitative methods*, ser. triangulation in action, 1979.

[110] Y. Jin, *Multi-objective machine learning*. Springer Science & Business Media, 2006, vol. 16.

[111] U. Johansson and L. Niklasson, "Evolving decision trees using oracle guides," in *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. IEEE, 2009.

[112] C. M. Judd, E. R. Smith, and L. H. Kidder, "Research Methods in Social Relations, Fort Worth: Holt, Rinehart and Winston," 1991.

[113] J. M. Juran, *Juran on planning for quality*. Collier Macmillan, 1988.

[114] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*. Springer Science & Business Media, Mar. 2013.

[115] A. Karwath and R. D. King, "Homology induction: the use of machine learning to improve sequence similarity searches," *BMC bioinformatics*, vol. 3, no. 1, p. 11, 2002.

[116] K. A. Kaufman and R. S. Michalski, "Learning from inconsistent and noisy data: the AQ18 approach," in *International Symposium on Methodologies for Intelligent Systems*. Springer, 1999, pp. 411–419.

[117] B. Kim, *Interactive and Interpretable Machine Learning Models for Human Machine Collaboration*. Massachusetts Institute of Technology, 2015.

[118] B. Kim, C. Rudin, and J. A. Shah, "The Bayesian Case Model - A Generative Approach for Case-Based Reasoning and Prototype Classification." *NIPS*, 2014.

[119] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Tech. Rep., 2007.

[120] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*. Springer Science & Business Media, Nov. 2007, pp. 63–92.

[121] H. K. Klein and M. D. Myers, "A set of principles for conducting and evaluating interpretive field studies in information systems," *MIS quarterly*, pp. 67–93, 1999.

[122] 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.

[123] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *2004 IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 2004, pp. 199–206.

[124] I. Kononenko, "Inductive and Bayesian learning in medical diagnosis," *Applied Artificial Intelligence an International Journal*, vol. 7, no. 4, pp. 317–337, 1993.

[125] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks." 2012.

[126] J. A. Krosnick, "Survey research," *Annual Review of Psychology*, vol. 50, no. 1, pp. 537–567, 1999.

[127] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, vol. cs.CV, 2016.

[128] G. Laforge, "Machine Intelligence at Google Scale," in *QCon*, Jun. 2018, pp. 1–58.

[129] H. Lakkaraju, S. H. Bach, and J. Leskovec, "Interpretable Decision Sets - A Joint Framework for Description and Prediction." *KDD*, pp. 1675–1684, 2016.

[130] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–17, Mar. 1977.

[131] F. Lau, "Toward a framework for action research in information systems studies," *Information Technology & People*, vol. 12, no. 2, pp. 148–176, 1999.

[132] N. Lavrač, "Selected techniques for data mining in medicine," *Artificial intelligence in medicine*, vol. 16, no. 1, pp. 3–23, 1999.

[133] T. Lei, R. Barzilay, and T. Jaakkola, "Rationalizing Neural Predictions," *arXiv.org*, Jun. 2016.

[134] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying Software Engineers: Data Collection Techniques for Software Field Studies," *Empirical Software Engineering*, vol. 10, no. 3, pp. 311–341, Jul. 2005.

[135] E. Lima, C. Mues, and B. Baesens, "Domain knowledge integration in data mining using decision tables: Case studies in churn prediction," *Journal of the Operational Research Society*, vol. 60, no. 8, pp. 1096–1106, 2009.

[136] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.

[137] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "How do api changes trigger stack overflow discussions? a study on the android sdk," in *proceedings of the 22nd International Conference on Program Comprehension*. ACM, 2014, pp. 83–94.

[138] Z. C. Lipton, "The Mythos of Model Interpretability." *IEEE Transactions on Evolutionary Computation*, 2016.

[139] M. S. Litwin and A. Fink, *How to measure survey reliability and validity*. Sage, 1995, vol. 7.

[140] Y. Liu, T. Kohlberger, M. Norouzi, G. E. Dahl, J. L. Smith, A. Mohtashamian, N. Olson, L. H. Peng, J. D. Hipp, and M. C. Stumpe, "Artificial Intelligence–Based Breast Cancer Nodal Metastasis Detection," *Archives of Pathology & Laboratory Medicine*, pp. arpa.2018–0147–OA–11, Oct. 2018.

[141] D. Lo Giudice, C. Mines, A. LeClair, R. Curran, and A. Homan, "How AI Will Change Software Development And Applications," Tech. Rep., Nov. 2016.

[142] W. Maalej and M. P. Robillard, "Patterns of Knowledge in API Reference Documentation," *International Software Engineering Research Network*, vol. 39, no. 9, pp. 1264–1282.

[143] S. MacDonell, M. Shepperd, B. Kitchenham, and E. Mendes, "How reliable are systematic reviews in empirical software engineering?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 676–687, Sep. 2010.

[144] T. E. Marshall and S. L. Lambert, "Cloud-based intelligent accounting applications: accounting task automation using IBM watson cognitive computing," *Journal of Emerging Technologies in Accounting*, vol. 15, no. 1, pp. 199–215, 2018.

[145] D. Martens, J. Vanthienen, W. Verbeke, and B. Baesens, "Performance of classification models from a user perspective," *Decision Support Systems*, vol. 51, no. 4, pp. 782–793, 2011.

[146] J. A. McCall, "Factors in software quality," *US Rome Air development center reports*, 1977.

[147] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in software quality. volume i. concepts and definitions of software quality," Tech. Rep., 1977.

[148] J. McCarthy, "Programs with Common Sense," Cambridge, MA, USA, Tech. Rep., 1960.

[149] L. McLeod and S. G. MacDonell, "Factors that affect software systems development project outcomes: A survey of research," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 24, 2011.

[150] J. Meltzoff, *Critical thinking about research: Psychology and related fields.* American psychological association, 1998.

[151] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri, "Challenges in software evolution," in *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, Sep. 2005, pp. 13–22.

[152] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, "Machine Learning and Statistical Classification of Artificial intelligence," 1994.

[153] D. Michie, "Machine Learning in the Next Five Years." *EWSL*, 1988.

[154] M. B. Miles, A. M. Huberman, M. A. Huberman, and M. Huberman, *Qualitative data analysis: An expanded sourcebook*. sage, 1994.

[155] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru, "Model cards for model reporting," *arXiv preprint arXiv:1810.03993*, pp. 220–229, 2018.

[156] D. L. Moody, "The "Physics" of Notations - Toward a Scientific Basis for Constructing Visual Notations in Software Engineering." *IEEE Trans. Software Eng.*, 2009.

[157] C. Murphy and G. E. Kaiser, "Improving the dependability of machine learning applications," 2008.

[158] C. Murphy, G. E. Kaiser, and M. Arias, "An approach to software testing of machine learning applications," 2007.

[159] B. A. Myers, S. Y. Jeong, Y. Xie, J. Beaton, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, "Studying the documentation of an API for enterprise Service-Oriented Architecture," *Journal of Organizational and End User Computing (JOEUC)*, vol. 22, no. 1, pp. 23–51, 2010.

[160] S. Nakajima, "Model-checking verification for reliable web service," in *OOPSLA 2002 Workshop on Object-Oriented Web Services, Seattle, Washington*, 2002.

[161] M. Narayanan, E. Chen, J. He, B. Kim, S. Gershman, and F. Doshi-Velez, "How do Humans Understand Explanations from Machine Learning Systems? An Evaluation of the Human-Interpretability of Explanation." *IEEE Transactions on Evolutionary Computation*, 2018.

[162] S. Narayanan and S. A. McIlraith, "Simulation, verification and automated composition of web services," in *Proceedings of the 11th international conference on World Wide Web*. ACM, 2002, pp. 77–88.

[163] B. J. Nelson, "Remote procedure call," Ph.D. dissertation, Carnegie Mellon University, 1981.

[164] Y. Nishi, S. Masuda, H. Ogawa, and K. Uetsuki, "A test architecture for machine learning product," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2018, pp. 273–278.

[165] N. Novielli, F. Calefato, and F. Lanubile, "The challenges of sentiment detection in the social programmer ecosystem," in *Proceedings of the 7th International Workshop on Social Software Engineering*. ACM, 2015, pp. 33–40.

[166] K. Nybom, A. Ashraf, and I. Porres, "A Systematic Mapping Study on API Documentation Generation Approaches," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Prague, Czech Republic, pp. 462–469.

[167] 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.

[168] T. Ohtake, A. Cummaudo, M. Abdelrazek, R. Vasa, and J. Grundy, "Merging Intelligent API Responses using a Proportional Representation Approach," in *International Conference on Web Engineering ICWE*, Daejeon, Korea, Jun. 2019, pp. 391–406.

[169] Open Software Foundation, "Part 3: DCE Remote Procedure Call (RPC)," in *OSF DCE application development guide: revision 1.0*. Prentice Hall, Dec. 1991.

[170] N. Oreskes, K. Shrader-Frechette, and K. Belitz, "Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences," *Science*, vol. 263, no. 5147, pp. 641–646, 1994.

[171] A. L. M. Ortiz, "Curating Content with Google Machine Learning Application Programming Interfaces," in *EIAPortugal*, Jul. 2017.

[172] F. E. Otero and A. A. Freitas, "Improving the interpretability of classification rules discovered by an ant colony algorithm," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 73–80.

[173] A. Pal, S. Chang, and J. A. Konstan, "Evolution of experts in question answering communities." in *ICWSM*, 2012.

[174] A. Parasuraman, V. A. Zeithaml, and L. L. Berry, "Servqual: A multiple-item scale for measuring consumer percptions of service quality," *Journal of retailing*, vol. 64, no. 1, pp. 12–29, 1988.

[175] R. Parekh, "Designing AI at Scale to Power Everyday Life," in *the 23rd ACM SIGKDD International Conference*. New York, New York, USA: ACM Press, 2017, pp. 27–27.

[176] C. Pautasso, O. Zimmermann, and F. Leymann, "**RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision** ," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 805–814.

[177] M. Pazzani, "Comprehensible knowledge discovery: gaining insight from data," in *First Federal Data Mining Conference and Exposition*, 1997, pp. 73–82.

[178] M. J. Pazzani, S. Mani, and W. R. Shankle, "Acceptance of rules generated by machine learning among medical experts," *Methods of information in medicine*, vol. 40, no. 05, pp. 380–385, 2001.

[179] J. Pearl, "The Seven Tools of Causal Inference with Reflections on Machine Learning," 2018.

[180] K. Petersen and C. Gencel, "Worldviews, Research Methods, and their Relationship to Validity in Empirical Software Engineering Research," in *2013 Joint Conference of the 23nd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement (IWSM-MENSURA)*. IEEE, Jan. 2019, pp. 81–89.

[181] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering." in *EASE*, 2008, pp. 68–77.

[182] Z. Pezzementi, T. Tabor, S. Yim, J. K. Chang, B. Drozd, D. Guttendorf, M. Wagner, and P. Koopman, "Putting image manipulations in context: robustness testing for safe perception," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2018, pp. 1–8.

[183] H. Pham, *Software reliability*. Springer Science & Business Media, 2000.

[184] M. Piccioni, C. A. Furia, and B. Meyer, "An Empirical Study of API Usability," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2013, pp. 5–14.

[185] R. M. Pirsig, *Zen and the art of motorcycle maintenance: An inquiry into values*. William Morrow and Company, 1974.

[186] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[187] J. R. Quinlan, "C4. 5: Programming for machine learning," *Morgan Kauffmann*, vol. 38, p. 48, 1993.

[188] ——, "Some elements of machine learning," in *International Conference on Inductive Logic Programming*. Springer, 1999, pp. 15–18.

[189] M. Rebouças, G. Pinto, F. Ebert, W. Torres, A. Serebrenik, and F. Castor, "An empirical study on the usage of the swift programming language," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*. IEEE, 2016, pp. 634–638.

[190] A. Reis, D. Paulino, V. Filipe, and J. Barroso, "Using Online Artificial Vision Services to Assist the Blind - an Assessment of Microsoft Cognitive Services and Google Cloud Vision." *WorldCIST*, vol. 746, no. 12, pp. 174–184, 2018.

[191] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?"," in *the 22nd ACM SIGKDD International Conference*. New York, New York, USA: ACM Press, 2016, pp. 1135–1144.

[192] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, "MLaaS: Machine Learning as a Service," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, Dec. 2015, pp. 896–902.

[193] G. Richards, V. J. Rayward-Smith, P. H. Sönksen, S. Carey, and C. Weng, "Data mining for indicators of early mortality in a database of clinical records," *Artificial intelligence in medicine*, vol. 22, no. 3, pp. 215–231, 2001.

[194] G. Ridgeway, D. Madigan, T. Richardson, and J. O'Kane, "Interpretable Boosted Naïve Bayes Classification." *KDD*, 1998.

[195] RightScale Inc., "RightScale 2018 State of the Cloud Report," Tech. Rep., 2018.

[196] G. Ritzer and E. Guba, "The Paradigm Dialog," *Canadian Journal of Sociology / Cahiers canadiens de sociologie*, vol. 16, no. 4, p. 446, 1991.

[197] S. P. Robbins and T. Judge, *Essentials of organizational behavior*. Pearson,, 2014.

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

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

[200] H. Robinson, J. Segal, and H. Sharp, "Ethnographically-informed empirical studies of software practice," *Information and Software Technology*, vol. 49, no. 6, pp. 540–551, 2007.

[201] L. Rokach and O. Z. Maimon, *Data mining with decision trees: theory and applications*. World scientific, 2008, vol. 69.

[202] C. Rosen and E. Shihab, "What are mobile developers asking about? A large scale study using stack overflow," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.

[203] W. Rosenberry, D. Kenney, and G. Fisher, *Understanding DCE*. O'Reilly & Associates, Inc., 1992.

[204] A. Rosenfeld, R. Zemel, and J. K. Tsotsos, "The elephant in the room," *arXiv preprint arXiv:1808.03305*, 2018.

[205] A. S. Ross, M. C. Hughes, and F. Doshi-Velez, "Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations," *arXiv.org*, Mar. 2017.

[206] R. J. Rubey and R. D. Hartwick, "Quantitative measurement of program quality," in *Proceedings of the 1968 23rd ACM national conference*. New York, New York, USA: ACM, 1968, pp. 671–677.

[207] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), ser. CRM Monograph Series. American Mathematical Society, 2004, vol. 23.

[208] H. W. Schmidt, I. Crnkovic, G. T. Heineman, and J. A. Stafford, Eds., *A Framework for Contract-Based Collaborative Verification and Validation of Web Services*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[209] M. Schwabacher, P. Langley, and P. Norvig, "Discovering communicable scientific knowledge from spatio-temporal data," *ICML*, 2001.

[210] C. B. Seaman, "Qualitative methods," in *Guide to Advanced Empirical Software Engineering*. Springer Science & Business Media, Nov. 2007, pp. 35–62.

[211] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 618–626.

[212] S. Sen and L. Knight, "A genetic prototype learner," in *IJCAI*. Citeseer, 1995, pp. 725–733.

[213] C. E. Shannon and W. Weaver, *The mathematical theory of communication*. Urbana, IL: The University of Illinois Press, 1963.

[214] M. Shaw, "Writing good software engineering research papers," in *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 2003, pp. 726–736.

[215] Shull, Forrest, Singer, Janice, and Sjøberg, Dag I K, *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer Science & Business Media, Nov. 2007.

[216] H. A. Simon, *The sciences of the artificial*. MIT press, 1996.

[217] J. Singer, S. E. Sim, and T. C. Lethbridge, "Software engineering data collection for field studies," in *Guide to Advanced Empirical Software Engineering*. Springer Science & Business Media, Nov. 2007, pp. 9–34.

[218] S. Singh, M. T. Ribeiro, and C. Guestrin, "Programs as Black-Box Explanations," *arXiv.org*, Nov. 2016.

[219] V. S. Sinha, S. Mani, and M. Gupta, "Exploring activeness of users in QA forums," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 77–80.

[220] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[221] I. Sommerville, *Software Engineering*, 9th ed. Addison-Wesley, 2011.

[222] P. E. Spector, *Summated rating scale construction: An introduction*. Sage, 1992.

[223] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks." *IEEE Transactions on Evolutionary Computation*, 2019.

[224] G. H. Subramanian, J. Nosek, S. P. Raghunathan, and S. S. Kanitkar, "A comparison of the decision table and tree," *Communications of the ACM*, vol. 35, no. 1, pp. 89–94, 1992.

[225] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live API documentation," in *the 36th International Conference*. New York, New York, USA: ACM Press, 2014, pp. 643–652.

[226] M. Sugiyama, N. D. Lawrence, and A. Schwaighofer, *Dataset shift in machine learning*. The MIT Press, 2017.

[227] N. R. Suri, V. S. Srinivas, and M. N. Murty, "A cooperative game theoretic approach to prototype selection," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2007, pp. 556–564.

[228] D. Szafron, P. Lu, R. Greiner, D. S. Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell, and A. Fyshe, "Proteome Analyst: custom predictions with explanations in a web-based tool for high-throughput proteome annotations," *Nucleic acids research*, vol. 32, no. 2, pp. W365–W371, 2004.

[229] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[230] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision." *25th IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[231] M. B. W. Tabor, "**Student Proves That S.A.T. Can Be: (D) Wrong**," *New York Times*, Feb. 1997.

[232] A. Tahir, A. Yamashita, S. Licorish, J. Dietrich, and S. Counsell, "Can you tell me if it smells?: A study on how developers discuss code smells and anti-patterns in Stack Overflow," *the 22nd International Conference*, pp. 68–78, 2018.

[233] G. Tassey, *The Economic Impacts of Inadequate Infrastructure for Software Testing*. National Institute of Standards and Technology, Sep. 2002.

[234] R. S. Taylor, "Question-negotiation and information-seeking in libraries (Vol. 29): College and Research Libraries," 1968.

[235] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolution using topic models," *Science of Computer Programming*, vol. 80, pp. 457 – 479, 2014.

[236] S. Thrun, "Is Learning The n-th Thing Any Easier Than Learning The First?" p. 7, 1996.

[237] Q. Tu *et al.*, "Evolution in open source software: A case study," in *Proceedings 2000 International Conference on Software Maintenance*. IEEE, 2000, pp. 131–142.

[238] M. Usman, R. Britto, J. Börstler, and E. Mendes, "Taxonomies in software engineering: A Systematic mapping study and a revised taxonomy development method," *Information and Software Technology*, vol. 85, pp. 43–59, May 2017.

[239] A. Van Assche and H. Blockeel, "Seeing the forest through the trees: Learning a comprehensible model from an ensemble," in *European conference on machine learning*. Springer, 2007, pp. 418–429.

[240] B. Venners, "Design by Contract: A Conversation with Bertrand Meyer," *Artima Developer*, 2003.

[241] W. Verbeke, D. Martens, C. Mues, and B. Baesens, "Building comprehensible customer churn prediction models with advanced rule induction techniques," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2354–2364, 2011.

[242] S. Wachter, B. Mittelstadt, and L. Floridi, "Why a Right to Explanation of Automated Decision-Making Does Not Exist in the General Data Protection Regulation," *International Data Privacy Law*, vol. 7, no. 2, pp. 76–99, Jun. 2017.

[243] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With Great Training Comes Great Vulnerability - Practical Attacks against Transfer Learning." *USENIX Security Symposium*, 2018.

[244] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in StackOverflow," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1019–1024.

[245] W. Wang, H. Malik, and M. W. Godfrey, "Recommending Posts concerning API Issues in Developer Q&A Sites," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 224–234.

[246] R. B. Watson, "Development and application of a heuristic to assess trends in API documentation," in *30th ACM international conference on Design of communication*. Seattle, Washington, USA: ACM, 2012, pp. 295–302.

[247] S. Weerawarana, *Web Services Platform Architecture*, ser. SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice-Hall PTR, 2005.

[248] D. Wettschereck, D. W. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 273–314, 1997.

[249] H. Wickham, "A Layered Grammar of Graphics," *Journal of Computational and Graphical Statistics*, vol. 19, no. 1, pp. 3–28, Jan. 2010.

[250] R. J. Wieringa and J. M. Heerkens, "The methodological soundness of requirements engineering papers: a conceptual framework and two case studies," *Requirements engineering*, vol. 11, no. 4, pp. 295–307, 2006.

[251] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[252] C. Wohlin and A. Aurum, "Towards a decision-making structure for selecting a research design in empirical software engineering," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1427–1455, May 2014.

[253] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[254] M. L. Wong and K. S. Leung, *Data mining using grammar based genetic programming and applications*. Springer Science & Business Media, 2006, vol. 3.

[255] X. Yi and K. J. Kochut, "A cp-nets-based design and verification framework for web services composition," in *Web Services, 2004. Proceedings. IEEE International Conference on*. IEEE, 2004, pp. 756–760.

[256] R. K. Yin, *Case study research and applications: Design and methods*. Sage publications, 2017.

[257] J. Zahálka and F. Železný, "An experimental test of Occam's razor in classification," *Machine Learning*, vol. 82, no. 3, pp. 475–481, 2011.

[258] J. Zhang and R. Kasturi, "Extraction of Text Objects in Video Documents: Recent Progress," in *2008 The Eighth IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 2008, pp. 5–17.

[259] B. Zupan, J. DemšAr, M. W. Kattan, J. R. Beck, and I. Bratko, "Machine learning for survival analysis: a case study on recurrence of prostate cancer," *Artificial intelligence in medicine*, vol. 20, no. 1, pp. 59–75, 2000.

[260] M. zur Muehlen, J. V. Nickerson, and K. D. Swenson, "Developing web services choreography standards—the case of REST vs. SOAP," *Decision Support Systems*, vol. 40, no. 1, pp. 9–29, Jul. 2005.

Online Artefacts

[261] C. Howard, "Introducing Google AI," https://ai.googleblog.com/2018/05/introducing-google-ai.html, May 2018.

[262] "Detect research-methodology | Google Cloud Vision API Documentation | Google Cloud," https://cloud.google.com/vision/docs/research-methodology, accessed: 28 August 2018.

[263] "Class EntityAnnotation | Google.Cloud.Vision.V1," https://googlecloudplatform.github.io/google-cloud-dotnet/docs/Google.Cloud.Vision.V1/api/Google.Cloud.Vision.V1.EntityAnnotation.html, accessed: 28 August 2018.

[264] "How to call the Computer Vision API," https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/vision-api-how-to-topics/howtocallvisionapi, accessed: 28 August 2018.

[265] "azure-sdk-for-java/ImageTag.java," https://github.com/Azure/azure-sdk-for-java/blob/8d70f41fbdb88b3a9297af5ba24551cf26f40ad4/cognitiveservices/data-plane/vision/computervision/src/main/java/com/microsoft/azure/cognitiveservices/vision/computervision/models/ImageTag.java#24, accessed: 28 August 2018.

[266] "Detecting research-methodology in an image," https://docs.aws.amazon.com/rekognition/latest/dg/research-methodology-detect-research-methodology-image.html, accessed: 28 August 2018.

[267] "Detecting objects and scenes," https://docs.aws.amazon.com/rekognition/latest/dg/research-methodology.html, accessed: 28 August 2018.

[268] "Vision API - Image Content Analysis Âă|Âă Cloud Vision API Âă|Âă Google Cloud," http://bit.ly/2TD9mBs, accessed: 13 September 2018.

[269] "Image Processing with the Computer Vision API | Microsoft Azure," http://bit.ly/2YqhkS6, accessed: 13 September 2018.

[270] "Amazon Rekognition," https://amzn.to/2TyT2BL, accessed: 13 September 2018.

[271] "Detecting labels in an image," http://bit.ly/2UlkW9K, accessed: 13 September 2018.

[272] "Watson visual recognition," https://ibm.co/2TBNIO4, accessed: 13 September 2018.

[273] "Image Recognition API & Visual Search Results," http://bit.ly/2UmNPCw, accessed: 13 September 2018.

[274] "Clarifai," http://bit.ly/2TB3kSa, accessed: 13 September 2018.

[275] "Image Recognition API | DeepAI," http://bit.ly/2TBNYgf, accessed: 26 September 2018.

[276] "Imagga - powerful image recognition apis for automated categorization & tagging in the cloud and on-premises," http://bit.ly/2TxsyRe, accessed: 13 September 2018.

[277] "Image recognition - talkwalker," http://bit.ly/2TyT7W5, accessed: 13 September 2018.

[278] "Megvii," http://bit.ly/2WJYFzk, accessed: 3 April 2019.

[279] "Tuputech," http://bit.ly/2uF4IsN, accessed: 3 April 2019.

[280] "Yitu technology," http://bit.ly/2uGvxgf, accessed: 3 April 2019.

[281] "Sensetime," http://bit.ly/2WH6RjF, accessed: 3 April 2019.

[282] “Detect Labels | Google Cloud Vision API Documentation | Google Cloud,” http://bit.ly/2TD5kcy, accessed: 28 August 2018.
[283] “Detecting labels in an image,” https://amzn.to/2TBNtTa, accessed: 28 August 2018.
[284] “How to call the Computer Vision API,” http://bit.ly/2TD5oJk, accessed: 28 August 2018.
[285] “Deepglint,” http://bit.ly/2uHHdPS, accessed: 3 April 2019.
[286] http://bit.ly/2KlyhcD, accessed: 27 March 2019.
[287] http://bit.ly/2G6ZOeC, accessed: 27 March 2019.
[288] http://bit.ly/2G7saFJ, accessed: 27 March 2019.
[289] “What is Computer Vision?” http://bit.ly/2TDgUnU, accessed: 28 August 2018.
[290] http://bit.ly/2G5ZEEe, accessed: 27 March 2019.
[291] K. Ballinger, “Simplicity and utility, or, why soap lost,” http://keithba.net/simplicity-and-utility-or-why-soap-lost, Dec. 2014.
[292] L. Mandel, “Describe REST Web services with WSDL 2.0,” https://www.ibm.com/developerworks/webservices/library/ws-restwsdl/, May 2008.

Part IV

Appendices

APPENDIX A

Additional Materials

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

The development of web 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 DCE introduced a collection of services and tools for developing and maintaining distributed systems using a client/server architecture [203]. This framework used the synchronous communication paradigm RPCs first introduced by Nelson [163] that allows procedures to be called in a remote address space as if it were local. Its communication paradigm, DCE/RPC [169], 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 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 [169]). As new functionality was introduced, this lead to the natural development of the SOAP, the backbone messaging connector for WS applications, a realisation of the SOA [37] pattern. The SOA pattern prescribes that services are offered by service providers and consumed by service consumers in a platform- and language-agnostic manner and are used in large-scale enterprise systems (e.g., banking, health). Key to the SOA pattern is that a service's quality attributes (see Section 2.1) can be specified and guaranteed using a SLA whereby the consumer and provider agree upon a set level of service, which in some cases are legally binding [15]. This agreement can be measured using 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 [99, 247]). These attributes are included within SOAP headers; thus, QOS aspects are independent from the transport layer and instead exist at the application layer [176]. The IDL of SOAP is WSDL, providing a description of how the web service is invoked, what parameters to expect, and what data structures are returned.

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

While it is rich in metadata and verbosity, discussions on whether this was a benefit or drawback came about the mid-2000s [176, 260] whether the amount of data transfer paid off (especially for mobile clients where data usage was scarce). Developer usability for debugging the SOAP 'envelopes' (messages POSTed over HTTP to the service provider component) was difficult, both due to the nature of XML's wordiness and difficulty to test (by sending POST requests) in-browser. As a simple example, 25 lines (794 bytes) of HTTP communication is transferred to request a customer's name from a record using SOAP (Listings A.1 and A.2).

Listing A.1: A SOAP HTTP POST consumer request to retrieve customer record #43456 from a web service provider. Source: [291].

```
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 Listing A.1. Source: [291].

```
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 [176]. That is, the HTTP is considered as a transport protocol solely. In 2000, Fielding [69] introduced REST, which instead approaches the web as a medium to publish data (i.e., HTTP is part of the *application* layer

instead). Hence, applications become amalgamated into of the Web. Fielding bases REST on four key principles:

- **URIs identify resources.** Resources and services have a consistent global address space that aides in their discovery via URIs [18].
- **HTTP verbs manipulate those resources.** Resources are manipulated using the four consistent CRUD verbs provided by HTTP: POST, GET, PUT, DELETE.
- **Self-descriptive messages.** Each request provides enough description and context for the server to process that message.
- **Resources are stateless.** Every interaction with a resource is stateless.

Consider the equivalent example of Listings A.1 and A.2 but in a RESTful architecture (Listings A.3 and A.4) and it is clear why this style has grown more popular with developers (as we highlight in Figure A.1). Developers have since embraced RESTful API development, though the major drawback of RESTful services is its lack of a uniform IDL to facilitate development (though it is possible to achieve this using WADL [292]). Therefore, no RESTful service uses a standardised response document or invocation syntax. While there are proposals, such as WADL [88], 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 Listing A.1, but using REST. Source: [291].

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

Listing A.4: The REST HTTP service provider response for Listing A.3. Source: [291].

```
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 AI-Based Products And Services Is Already Visible. (From [141].)

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 these intelligent cloud services.

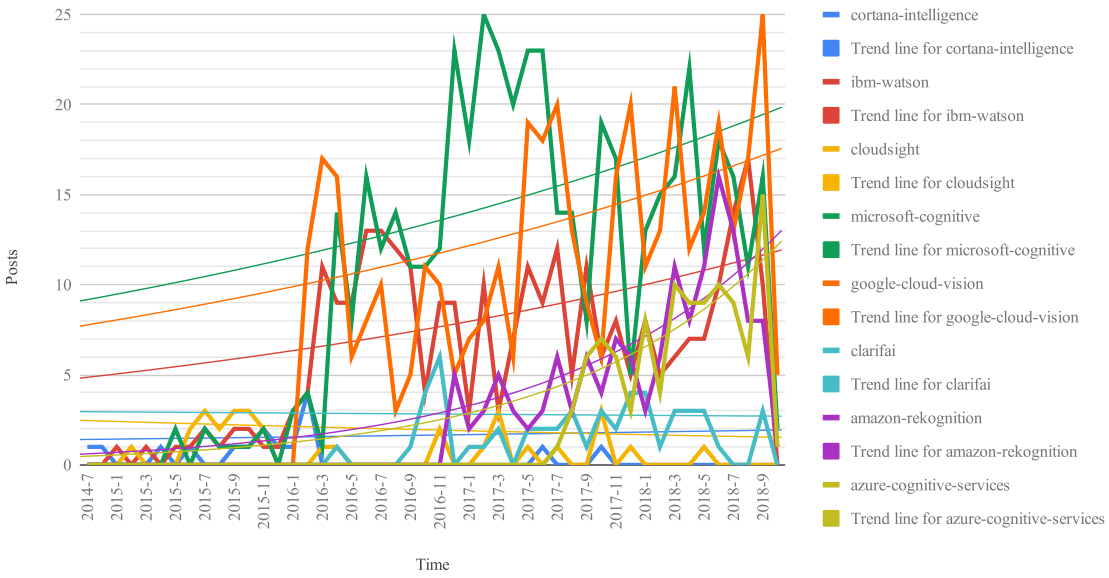


Figure A.4: Questions asked by software engineering researchers (column 2) that can be answered by field study techniques. (From [217].)

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

DRAFT: appendix/authorship-statements/icsme2019 will be inserted here DRAFT: appendix/authorship-statements/esem2019 will be inserted here DRAFT: appendix/authorship-statements/icwe2019 will be inserted here

APPENDIX C

Ethics Clearance

DRAFT: appendix/ethics/1-recommendations will be inserted here DRAFT: appendix/ethics/2-factors will be inserted here