A framework to document Cloud Intelligence Services using insights from Software Engineers

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

Confirmation of Candidature Report

Supervisory Panel:

Prof. Rajesh Vasa <rajesh.vasa@deakin.edu.au>
Prof. John Grundy <john.grundy@monash.edu>
A/Prof. Mohamed Abdelrazek <mohamed.abdelrazek@deakin.edu.au>
A/Prof. Andrew Cain <andrew.cain@deakin.edu.au>

Applied Artificial Intelligence Institute
Deakin University
Melbourne, Australia

January 16, 2019

Contents

List of Figures

List of Tables

List of Listings

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 [116] as replicated in Figure 1.1. Moreover, developers eager to develop a next generation of software are shifting away from mobile-first to 'AI-first' apps, that will reason, sense, think, act, listen, speak and execute our whims right within the palms of our hands. A wave of AI-first thinking embedded in companies' product lines is spearheaded through work achieved at Google, Microsoft and Facebook; for instance, Google's 2018 rebranding of *Google Research* to *Google AI* [82] or how AI is leveraged *at scale* within Facebook's infrastructure and platforms to serve its users with an AI-first attitude [131].

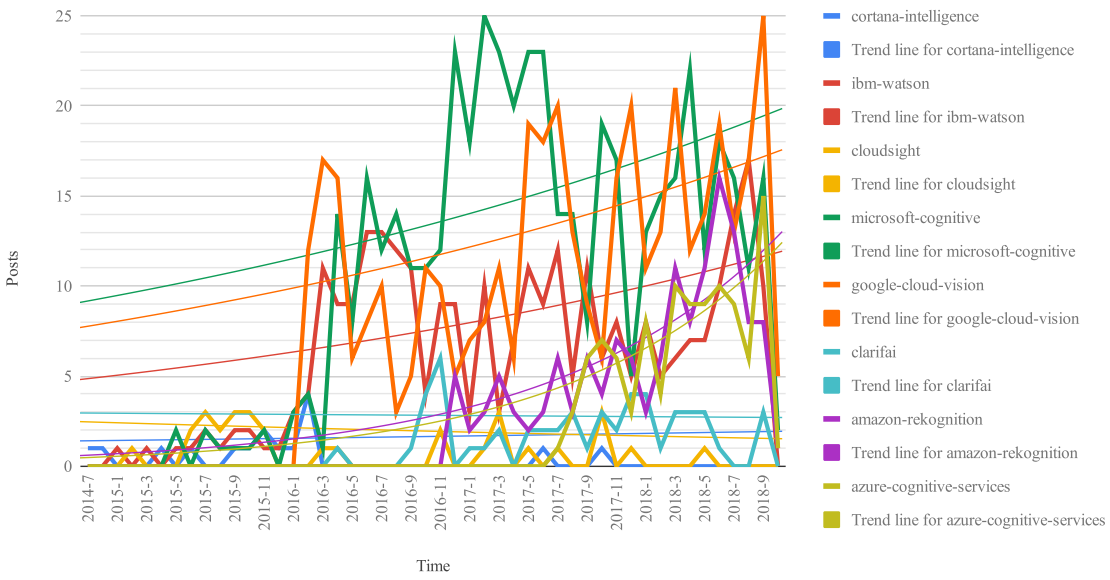
These services aim to lower the entry barrier to develop, test and deploy AI-first software in both skill and time. Software engineers needn't require a formal training in machine-learning nor a strong understanding of mathematics: thus, *skill required* is reduced. The training of such classifiers involves the laborious process of sourcing, curating and labelling large datasets: using such services does not, and thus *time* is reduced. To this end, they needn't require much machine-learning expertise or experience at all; instead, the process is abstracted behind an API call, only requiring knowledge on how to use a RESTful architecture [67] to access the cloud service.

To contrast this with more traditional means, a developer may choose to write up a deep-learning NN (for example) and train it using their own dataset. While this is laborious in time and demands significant knowledge in machine learning, the developer has full control over the models she creates. Alternatively, she may choose to download a pre-trained model and ML framework, such as Tensorflow [24]; less demanding in time but still requiring the knowledge to wire-up models with frameworks.

Figure 1.1: A Broad Range of AI-Based Products And Services Is Already Visible. (From [116].)

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 1.2: Number of posts categorised on Stack Overflow under popular computer vision cloud intelligence services.



This concept was coined by Google AI over various presentations as *The Machine Learning Spectrum* [128, 108, 20], which encompasses the variety of skill, effort, target audiences and outputs of various ML products. On one extreme is the research of developing algorithms (e.g., neural networks) to achieve intelligence, produced chiefly in academia—coined as BYOML [128, 20, 17]. 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 off-cloud solutions that assist in turning custom datasets into some form of predictive intelligence. We illustrate this further within **??**.

⟨ TODO: *Insert figure here about the 'levels' of AI including CIS - refer to RV's diagram.* ⟩

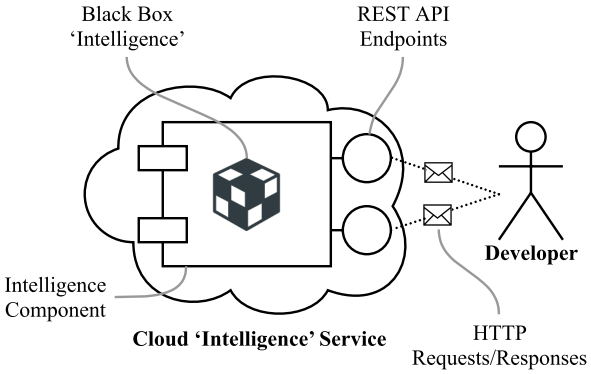
With less time and skill required to build AI-first apps using these 'friendly' ML services, these services have begun to gain traction within developer circles: Figure 1.2 shows the increasing trend of posts since 2014 on Stack Overflow that categorise popular computer vision cloud APIs.[1] A growing popularity into such 'off-the-shelf', pre-packaged ML APIs sparked varied nomenclature in academia: *Cognitive Applications* and *Machine Learning Services* [85] or *Machine Learning as a Service* [144] are some coined phrases in which the intelligence is provided or created as a service via the cloud. Some of these services provide the infrastructure to rapidly begin training from custom datasets (Google's AutoML[2] 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 'Cloud Intelligence

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

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

Services' (CISs), and diagrammatically express their usage within Figure 1.3.

Figure 1.3: Overview of Cloud Intelligence Services.



The general workflow of a CIS is relatively 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 [19] or ScaleAPI [22]. Thus, a CIS may be powered by varying degrees of intelligence: human intelligence, machine learning, data mining or even intelligence by brute-force.

While there are many 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 CVCISs [14, 7, 3, 21, 15, 9, 8, 11, 16, 23, 18, 10, 4]. The ubiquity of CVCISs is exemplified through evermore growing applications that use these APIs: aiding the vision-impaired [142, 54], accounting [117], data analytics [92], and student education [57]. Moreover, we refer to its growing adoption in developer circles within Figure 1.2.

1.1 Motivation: Current Developer Mindsets'

Figure 1.2 shows an increasing trend to the adoption and discussion of CISs with developers. As aforementioned, 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 to outputs (or response) without any understanding of the internal architecture by which this transformation occurs; indeed, this well-understood theory arose from the electronic sciences and since adapted to wider applications since the 1950s–60s [27, 41] to describe "systems whose internal mechanisms are not fully open to inspection" [27].

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 [134] 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 as traditionally their mindset is that computers will always make certain outcomes. But in the day and age of stochastic and probabilistic systems, this mindset needs to shift.

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 stochastic 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 [137, 148] written by providers, while providers should consider mismatches between the developer's conceptual knowledge of the API its implementation [105]. 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 [34] 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 1.1) and as is developer uptake and enthusiasm in the software engineering community (Figure 1.2). 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 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 wiHes 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).

Table 1.1: 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

More concretely, in Table 1.1, 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[3] to three different in August 2018 and January 2019, he would find that each provider is different in both the vocabulary used between them and the confidence values and labels within the *same* provider vary within a matter of five months. But what does a change in confidence mean? The tautological nature of the definition of these changing confidence values (as presented in the API documentation) provides no insight for Tom to understand why

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

there a change in confidence, which we show in Table 1.2, 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 explicity told.

Table 1.2: 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)."* [12]
	"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]." [13]
	"The overall score of the result. Range [0, 1]" [13]
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."* [5]
	"The level of confidence the service has in the caption." [6]
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."* [1]
	"[Provider C] also provide[s] a percentage score for how much confidence [Provider C] has in the accuracy of each detected label." [2]

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 his 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 many 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?

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 framework to assist his 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 a few 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., [115, 78, 64]), 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 [94, 47]; 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 regions of interest 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.

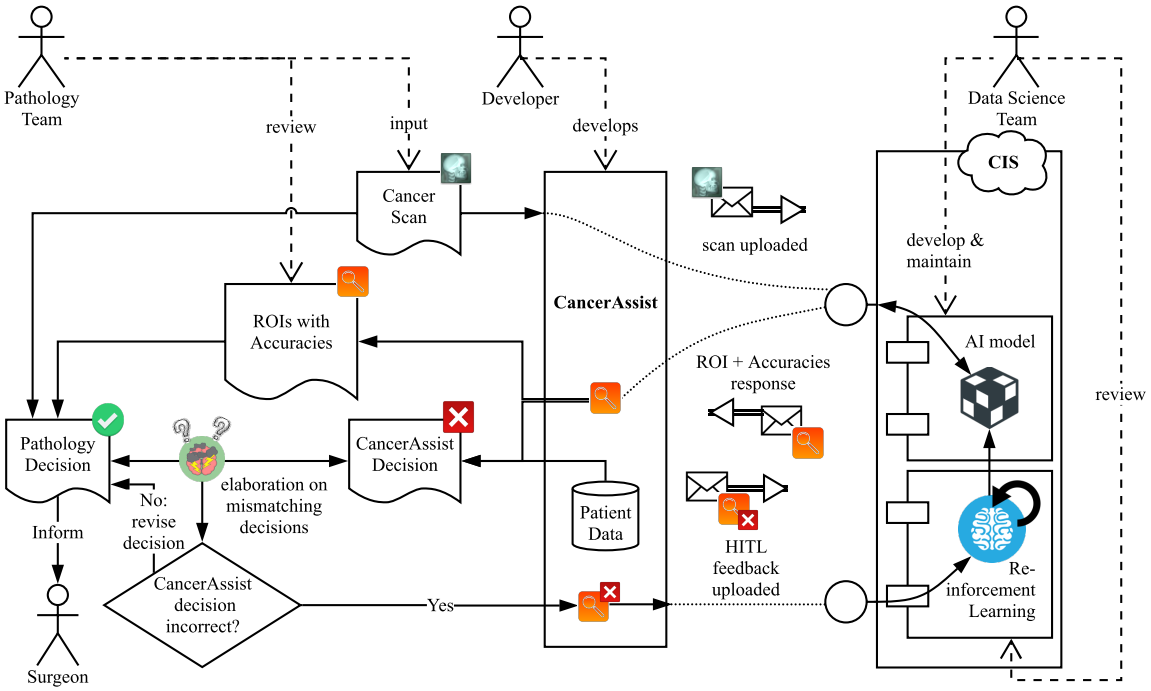


Figure 1.4: CancerAssist Context Diagram

Powering CancerAssist is Google AI's Lymph Node Assistant (LYNA) [115], a CNN based on the Inception-v3 model [167, 107]. 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.

Several 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 Data Science 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 Data Science team in training the model are correct?

Thus, it may be the case the improved documentation to communicate with the CIS and additional metadata is needed in the response object may prove useful. 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 best practices towards the usage of CISs to provide a guiding framework and recommendations for software 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 [157, 158].

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.

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

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.

Research Method Problem identification and discovery to validate our hypothesis in the *general context* of API usage will begin with a background to help inform what prior work has been done in the API documentation space. We will follow this with repository and question mining, i.e., searching on developer communities such as Quora, Stack Overflow, and GitHub Issues to find out what developers complain about and mine this knowledge into a framework.

We then will conduct an initial pre-controlled survey within our research group (we refer to as the 'pilot' survey study) and will use findings from the background and mining to help inform us of the kinds of questions to ask.

Findings from the pilot survey to help inform a wider structured survey and unstructured interview, where we will recruit external software engineers in industry through contacts of our research group. A quantitative (survey) and qualitative (interview) analysis will help begin to shape our research outcome of an API documentation quality assessment framework and help stabilise a general understanding of how developers use the existing APIs.

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?

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*".

Research Method 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.

Thus, we will use context-specific action research to develop basic 'prototypes' of varying contexts to help identify where any potential gaps are in the findings of RH1. To validate the findings of developer opinion in the surveys and interviews of RH1 are indeed genuine, this helps ensure that there is nothing missing by adding in further context to such opinions.

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.

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

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.

Research Method To confirm that the framework is valid, we will provide improved documentation and metadata responses of a set of popular CVCISs that is documented with the additional metadata and information organised using our framework.

We then ask 20 developers 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 and quantitative analysis.

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.

Chapter 2

Background

In Chapter 1, we defined a common set of 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 [14], AWS Rekognition [3], Azure Computer Vision [7], Watson Visual Recognition [15] 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 development of probabilistic software? What can we learn from common software engineering practices (e.g., [139, 162]) that apply to resolve this mismatch? 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 between CIS providers and consumers, the nature of applications built using 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.

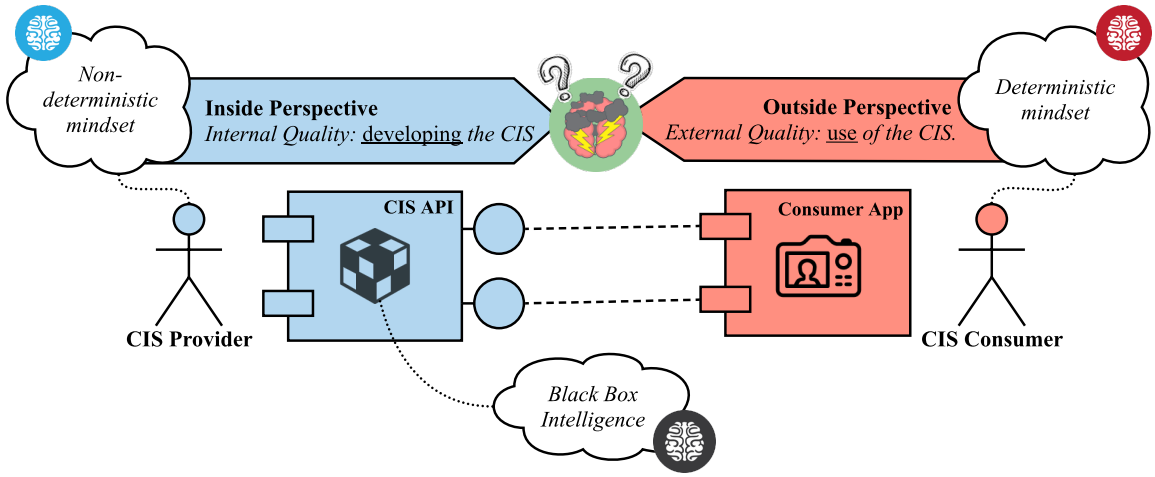


Figure 2.1: The three pillars by which we anchor the background: (1) developing a CIS with a non-deterministic mindset by the CIS provider; (2) the use of a CIS with a deterministic mindset by the CIS consumer; (3) the nature of a CIS itself.

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 [138]

The philosophical viewpoint of 'quality' remains highly debated and there are multiple facets to perceive this complex concept [74]. Transcendentally, a viewpoint like that of Pirsig's quote above shows that quality is not tangible but still recognisable; it's hard to explicitly define but you know when it's missing. Pragmatically, the International Organization for Standardization provides a breakdown of seven universally-applicable principles that defines quality for organisations, developers, customers and training providers [89]. More pertinently, though, the since withdrawn 1986 standard for quality was simply "the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs" [88].

Using this sentence, what characteristics exist for non-deterministic systems 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' [75, 53], 'meeting or exceeding customer expectation' [130], or 'fitness for use' [98]—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 [139] adapted the manufacturing-oriented view of quality from [32] and phrased software quality under three core pillars:

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

In the context of a non-deterministic CIS, however, are any of the above actually guaranteed? Given that the core of a system built using on top of 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

In his works on software reliability [136], Pham recounts the tale of a high-school student who sat a standardised test send out to 350,0000 students [168]. In the multiple-choice mathematical problem, the examiners used an algebraic equation using the letter a and intended that students *assume* that a was positive. The student, assuming that a could also be negative, answered the 'incorrect' choice of D instead of the 'correct' choice of C. After contacting the examiners to point out the flaw that *both* answers were indeed correct, up to 45,000 students had their

scores retrospectively boosted by up to 30 points. However, by the time the score alteration was made, students had already been admitted to a university (or not), and some suggested that a 10 point difference in score can alter the outcome of a university admittance or scholarship. The outcomes of a student's higher education were, thereby, affected by this one oversight in quality assessment.

So, it seems, the examiners had mislead students to answer 'incorrectly', leading to a poor question being written, and poor process standards to check if their 'correct' answers were indeed 100% correct. In the words of Boehm [36], the examiners "didn't build the right product" (exam) to effectively examine students, nor did they "build the product right" by failing to ensure quality standards were in their processes.

This story analogously describes the issues with the cost of quality [35] 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 [139], data sourced from Cigital Inc. [49] 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 [169].

Formally, we refer to the IEEE Standard Glossary of Software Engineering Terminology [86] 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 [162]. Though it is commonplace for providers to have a 'ship-first-fix-later' mentality of 'good-enough' software [172], 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; 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?

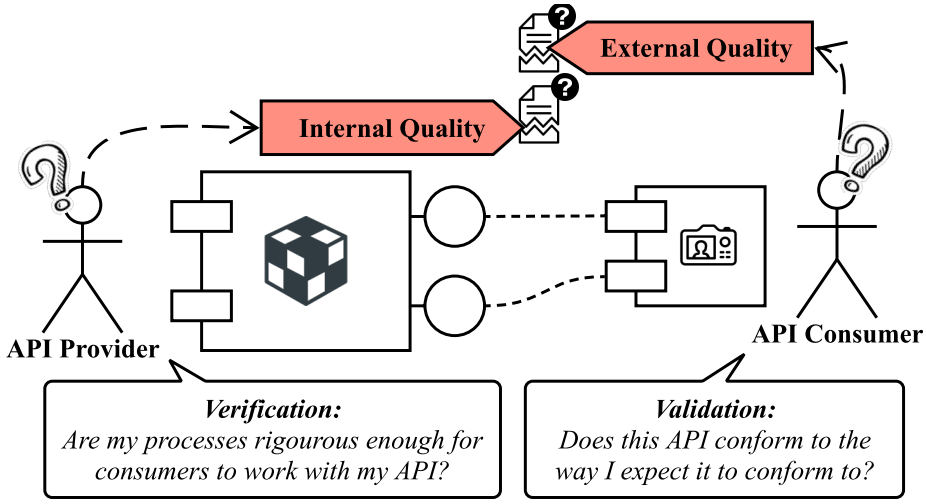


Figure 2.2: The 'leakage' of internal quality into the API consumer's product and external quality imposing on the API provider.

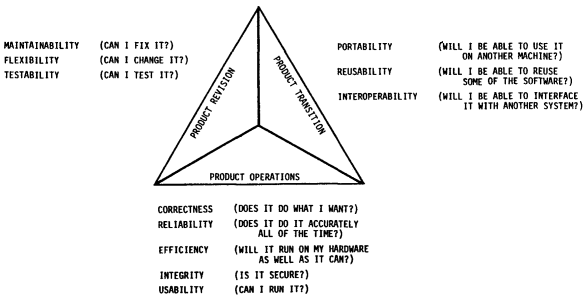
Therefore, just as the high-school student's test answers in the example we opened with are both correct and incorrect at the same time, so do CISs in returning a probabilistic result: no result is certain. While VV has been investigated in the area of mathematical and earth sciences for numerical probabilistic models and natural systems [127, 152], from the software engineering literature, little work has been achieved to look at the surrounding area of probabilistic systems hidden behind API calls.

Now that a developer is using a probabilistic system behind a deterministic API call, what does it mean in the context of VV? Do the current verification approaches and tools do 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 clearly 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

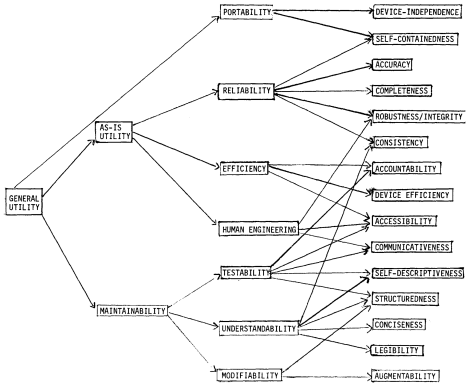
data was X), how can we achieve similar outputs when the system is not non-deterministic? A key problem with the probabilistic mindset is that once a model is 'fixed' by retraining it, while one data-point may be fixed, others may now have been effected, thereby not ensuring 100% validation. Thus, due to the unpredicatable and blurry nature of probabilistic systems, VV must be re-thought out extensively.

2.1.2 Quality Attributes and Models

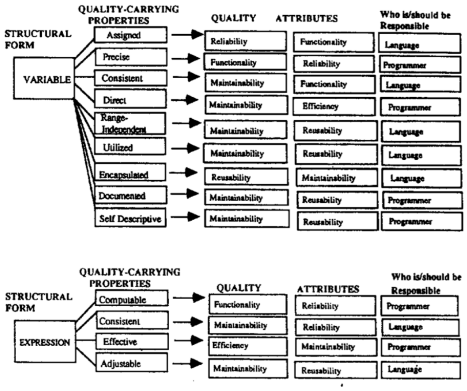
As we follow on from VV, we investigate similar approaches to the quality models that are used to try and capture some of the internal and external quality attributes via measurable metrics. There is no 'one' definition of quality and different perspectives on the issue has lead to different users placing varying value on disparate attributes.



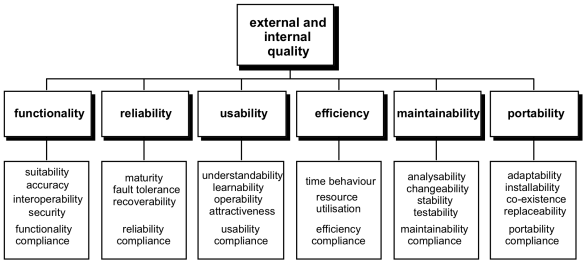
(a) McCall's quality software factors (1977) [119].



(b) Bohem's software quality characteristics tree (1978) [37].



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



(d) ISO/IEC software product evaluation characteristics (1999) [90].

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

Quality attribute assessment models are an early concept in software engineering, and systematically evaluating software quality appears as early as 1968 [151]. This study introduced 'attributes' as a "prose expression of the particular quality of desired software" (as

worded by Boehm et al. [37]) 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 [119, 46]. This model described quality from the three perspectives of product revision (*how can we keep the system operational?*), transition (*how can we migrate the system as needed?*) and operation (*how effective is the system at achieving its tasks?*) (Figure 2.3a). The model also introduced 11 attributes alongside numerous direct and indirect measures to help quantify quality. This model was further developed by Boehm et al. [37] who independently developed a model resembling McCall's, starting from an initial set of 11 software characteristics similar to that of McCall's but then diving deeper by defining 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 [90] (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 [87], 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 [25].

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 largely the same among all quality models:

McCall et al. Extent to which a program can be expected to perform its intended function with required precision [120].

Boehm et al. Code possesses the characteristic *reliability* to the extent that it can be expected to perform its intended functions satisfactorily [37].

Dromey Functionality implies reliability. The reliability of software is therefore largely 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 per-

formance when used under specified conditions [90].

These definitions strongly relate to the solution description (Section 2.1.3) in that reliability is the ability to maintain 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? 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 totally dependent on probabilistic outcomes. We explore this concept in further detail within the following subsection.

2.1.3 Solution Description

2.2 Probabilistic and Stochastic Systems

⟨ TODO: *What are stochastic/probabilistic systems? E.g., model interpretation?* ⟩

⟨ TODO: *What understanding might be missing from model interpretation? Relate back to topic.* ⟩

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 [141] and Michie [124]), and although there has since been a significant body of work in the area [160, 29, 143, 42, 150, 114, 38, 97, 28, 73, 55, 171, 31, 66, 113, 118, 132, 173], it is evident that 'accuracy' or model 'confidence' is still used as a primary criterion for AI evaluation [83, 93, 161]. Indeed, 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" [69], a problem commonly referred to as the *datashift* problem [164]. Analogously, Freitas [69] 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 very 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.*** *[69]*

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 [114]), what is known is (i) there exists a critical trade-off between accuracy and interpretability [68, 96, 101, 77, 59, 181], and (ii) a single quantifiable value cannot satisfy the subjective needs of end-users [69]. 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 [45, 102, 84]), there is often a mismatch between the formal objectives of the model (e.g., to minimise error) and complex real-world goals, where many 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" [126] to end-users. **Therefore, when human-decision makers must be interpretable themselves [146], 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. [61] discuss when explanations are not provided from a legal stance—for instance, those affected by algorithmic-based decisions have a 'right to explanation' [76, 174] under the European Union's GDPR[3]. But, explanations are not the only way to ensure

[1] In areas such as medicine [30, 110, 133, 145, 182, 171, 97, 65, 180, 94, 42], bioinformatics [70, 166, 100, 58, 95], finance [29, 84, 56] and customer analytics [173, 113].

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

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

AI accountability: theoretical guarantees (mathematical proofs) or statistical evidence can also serve as guarantees [61], however, in terms of explanations, what form they take and how they are proven correct are still open questions [114].

2.2.2 Explanation and Communication

From a SE 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' [121] 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 [156] applies, just as others have done with similar issues in the SE realm [125, 176] (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 [61] and thusly 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 2.4.

2.2.3 Mechanics of Model Interpretation

How do we interpret models? Methods for developing interpretation models include: decision trees [40, 79, 51, 140, 149], decision tables [113**?**] and decision sets [109, 126]; input gradients, gradient vectors or sensitivity analysis [154, 143, 111, 150, 29]; exemplars [103, 71]; generalised additive models [45]; classification (*if-then*) rules [170, 39, 50, 129, 178] and falling rule lists [160]; nearest neighbours [118, 155, 165, 175**?**] and Naïve Bayes

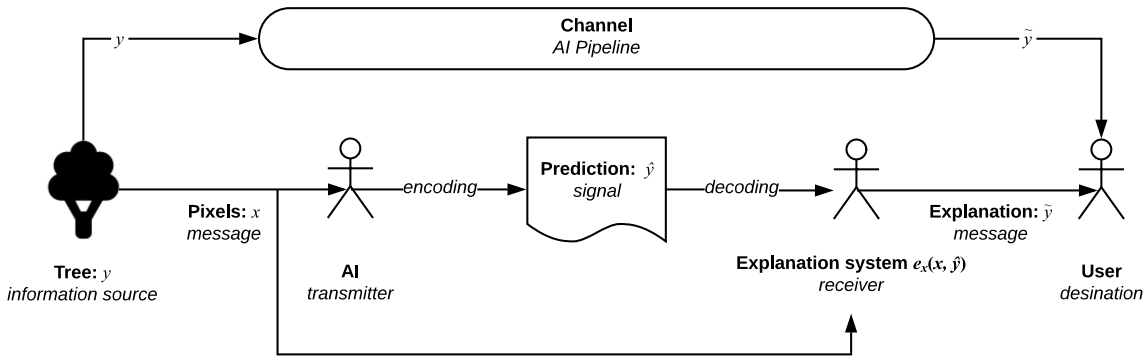


Figure 2.4: Theory of AI communication from information source, y, to intended user as explanations $\tilde{y}$.

analysis [30, 110, 106, 182, 123, 72, 48, 81]. Several cross-domain studies have assessed the interpretability of these techniques against end-users, measuring response time, accuracy in model response and user confidence [84, 80, 26, 163, 153, 70, 118, 173], although it is generally agreed that decision rules and decision tables provide the most interpretation in non-linear models such as SVMs or NNs [70, 118, 173]. For an extensive survey of the benefits and fallbacks of these techniques, we refer to Freitas [69] and Doshi-Velez and Kim [60].

As it stands, AI presents an issue with. (For a detailed discussion, see Doshi-Velez et al. [61].

2.3 Application Programming Interfaces

⟨ TODO: *History on APIs; i.e., original usage* ⟩

2.3.1 Documentation Standards

2.3.2 Web Services

2.3.3 RESTful APIs

⟨ TODO: *What are API documentation standards? What do they advocate for?* ⟩

⟨ TODO: *What is missing for AI documentation? What is the gap?* ⟩

2.4 Meta-modelling

⟨ TODO: *What is meta-modelling? Can get this from honours thesis...?* ⟩

To understand the methodology on how we captured our dataset, we must first introduce the three key notions behind MDE: technical spaces, models and systems. A system is a concrete "group or set of related or associated elements perceived or thought of as a unity or complex whole" [**?**]. Technical spaces were introduced by Ivanov et al. [91] as a model management framework based on algebraic structures (e.g., trees, (hyper)graphs, and categories). Technical spaces are usually based on a three-tier conjecture: meta-meta-models, meta-models and models. Whereas a model is an abstract representation of a concrete system of specific purpose, a *meta*-model, in contrast, describes the way to describe those models. A *meta-meta*-model can be used to describe the representation structure of our meta-models and defines a type system [44] that supports all underlying layers [33]. Figure 2.5 captures these concepts in further detail.

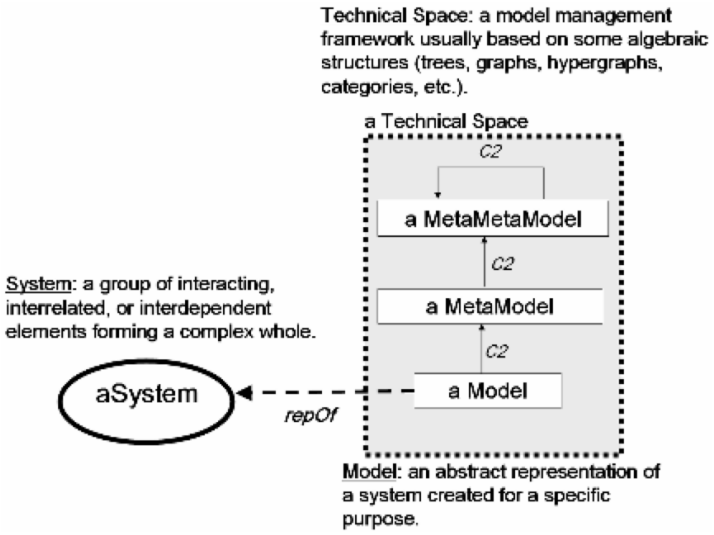


Figure 2.5: Systems, models and technical spaces. (From [33].)

⟨ TODO: *How does this differ in AI context?* ⟩

Chapter 3

Research Strategy

Investigating software engineering practices is oft 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 the research design utilised in this study by exploring six 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, (v) discuss practical considerations of our methods, and lastly (vi) explain a theory to review our data and relate it other studies in literature and our research questions. The foundations for developing this research strategy is expanded from that proposed by Easterbrook et al. [63].

3.1 Research Questions Revisited

In Section 1.2, we introduce three hypothesis of this study (RH1–3), 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 [122] 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 that focuses on designing better ways to approach software engineering tasks [158]. 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 [177].

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 [147]. In software engineering, four dominant philosophical stances are commonly characterised [52, 135]: 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 quite 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 [104]. 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 [43]. 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 [52] 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 [159]. The mixed-methods approach combines five classes of field study methods (or empirical strategies/studies) most relevant in empirical software engineering research [63, 179, 99]: 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. **[cite:Brathall and Jorgensen 2002]**, 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 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 several clearly defined hypotheses (RH1–3), 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 few 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 **[cite:Yin:2002]**. 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 criticial 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 2 (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 **[cite:Robinson:2007]**. 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 few 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 **[cite:Davidson:2004]** by actively seeking to intervene in the situation for the purpose of improving it. A precondition is to engage with a *problem owner* who is willing to collaborate in identifying and solving the problem faced. The problem must be authentic (a problem worth solving) and must have new knowledge outcomes for those involved. It is also characterised as an iterative approach to problem solving, where the knowledge gained from solving the problem has a desirable solution that empowers the problem owner and researcher.

This research is most associated to our adopted philosophical stance of critical theory. As this project is being conducted under the Applied Artificial Intelligence Institute 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 [159, 112] 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 3.1.

3.3.3 Experiment I: Qualitative mining of CIS API usage

Our first experiment aims to help shape the problem discovery in a generalised context on the existing usage of CIS APIs. Briefly, this experiment is comprised under two phases of survey research: (i) repository mining developer discussion forums (namely Stack Overflow); (ii) conducting unstructured interviews and distributing a questionarre (with a pre-trial experiment).

⟨ TODO: *Discuss how I will conduct the experiment* ⟩

Relevance and Motivation

⟨ TODO: *Relate to back to research hypotheses* ⟩

Data Collection & Analysis

⟨ TODO: *Discuss data collection techniques* ⟩

⟨ TODO: *Discuss data analysis techniques* ⟩

3.3.4 Experiment II: Observation of an improved CIS API

Action research

Overview

⟨ TODO: *Discuss how I will conduct the experiment* ⟩

Relevance and Motivation

⟨ TODO: *Relate to back to research hypotheses* ⟩

Data Collection & Analysis

⟨ TODO: *Discuss data collection techniques* ⟩

⟨ TODO: *Discuss data analysis techniques* ⟩

Figure 3.1: Questions asked by software engineering researchers (column 2) that can be answered by field study techniques. (From [159].)

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.

3.4 Empirical Validity

⟨ *TODO: Discuss empirical validity of both methods* ⟩

Threats to Validity...

Chapter 4

Project Status

4.1 Completed Work

4.2 Impact

4.3 Timeline

Chapter 5

Conclusion

References

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

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

[3] "Amazon Rekognition," https://aws.amazon.com/rekognition, accessed: 13 September 2018.

[4] "Home - affectiva : Affectiva," https://www.affectiva.com, accessed: 15 October 2018.

[5] "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.

[6] "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.

[7] "Image Processing with the Computer Vision API | Microsoft Azure," https://azure.microsoft.com/en-au/services/cognitive-services/computer-vision/, accessed: 13 September 2018.

[8] "Clarifai," https://www.clarifai.com, accessed: 13 September 2018.

[9] "Image Recognition API & Visual Search Results," https://docs.aws.amazon.com/rekognition/latest/dg/labels-detect-labels-image.html, accessed: 13 September 2018.

[10] "The face recognition company - cognitec," http://www.cognitec.com, accessed: 15 October 2018.

[11] "Image recognition api | deepai," https://deepai.org/ai-image-processing, accessed: 26 September 2018.

[12] “Detect Labels | Google Cloud Vision API Documentation | Google Cloud,” https://cloud.google.com/vision/docs/labels, accessed: 28 August 2018.

[13] “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.

[14] “Vision API - Image Content Analysis Âă|Âă Cloud Vision API Âă|Âă Google Cloud,” https://cloud.google.com/vision/, accessed: 13 September 2018.

[15] “Watson visual recognition,” https://www.ibm.com/watson/services/visual-recognition/, accessed: 13 September 2018.

[16] “Imagga - powerful image recognition apis for automated categorization & tagging in the cloud and on-premises,” https://imagga.com, accessed: 13 September 2018.

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

[18] “Kairos: Serving businesses with face recognition,” https://www.kairos.com, accessed: 15 October 2018.

[19] “Amazon Mechanical Turk,” https://www.mturk.com, accessed: 15 October 2018.

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

[21] “Detecting labels in an image,” https://docs.aws.amazon.com/rekognition/latest/dg/labels-detect-labels-image.html, accessed: 13 September 2018.

[22] “Scale: API for Training Data,” https://www.scaleapi.com, accessed: 15 October 2018.

[23] “Image recognition - talkwalker,” https://www.talkwalker.com/image-recognition, accessed: 13 September 2018.

[24] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[25] 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.

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

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

[28] 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.

[29] 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.

[30] 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.

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

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

[33] J. Bézivin, "Model Driven Engineering: An Emerging Technical Space," in *Generative and Transformational Techniques in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 36–64.

[34] 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.

[35] 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.

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

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

[38] 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.

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

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

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

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

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

[44] L. Cardelli and P. Wegner, "On understanding types, data abstraction, and polymorphism," *ACM Computing Surveys (CSUR)*, vol. 17, no. 4, pp. 471–523, Dec. 1985.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

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

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

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

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

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

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

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

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

[61] 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.

[62] R. G. Dromey, "A model for software product quality," *IEEE Transactions on Software Engineering*, 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] 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.

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

[68] 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.

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

[70] A. A. Freitas, D. C. Wieser, and R. Apweiler, "On the importance of comprehensi-

ble classification models for protein function prediction," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 7, no. 1, pp. 172–182, 2010.

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

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

[73] 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.

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

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

[76] B. Goodman and S. R. Flaxman, "EU regulations on algorithmic decision-making and a "right to explanation"." *CoRR*, 2016.

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

[78] 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.

[79] 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.

[80] 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.

[81] 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.

[82] C. Howard. (2018, May) Introducing Google AI. [Online]. Available: https://ai.googleblog.com/2018/05/introducing-google-ai.html

[83] 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.

[84] 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.

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

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

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

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

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

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

[91] I. Ivanov, J. BÃľzivin, and M. Aksit, "Technological spaces: An initial appraisal," 10 2002, pp. 1–6, <http://www.cs.rmit.edu.au/fedconf/2002/program.html>.

[92] 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.

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

[94] 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.

[95] 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.

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

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

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

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

[100] 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.

[101] 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.

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

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

[104] 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.

[105] 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.

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

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

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

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

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

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

[112] 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.

[113] 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.

[114] Z. C. Lipton, "The Mythos of Model Interpretability." *CoRR*, 2016.

[115] 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.

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

[117] 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.

[118] 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.

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

[120] 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.

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

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

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

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

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

[126] 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." *CoRR*, 2018.

[127] 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.

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

[129] 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.

[130] 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.

[131] 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.

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

[133] 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.

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

[135] 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.

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

[137] 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.

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

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

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

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

[142] 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.

[143] 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.

[144] 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.

[145] 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.

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

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

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

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

[150] 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.

[151] 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.

[152] 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.

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

[154] 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.

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

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

[157] 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.

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

[159] 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.

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

[161] 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.

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

[163] 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.

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

[165] 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.

[166] 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.

[167] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision." *CVPR*, 2016.

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

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

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

[171] 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.

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

[173] 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.

[174] 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.

[175] 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.

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

[177] 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.

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

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

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

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

[182] 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.

Appendix A

Submitted Ethics Application

Appendix B

Technical Report Manuscript