Don't Forget the Developer: A tale of Software Quality Operationalisation for Cloud Intelligence Services

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

October 26, 2018

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

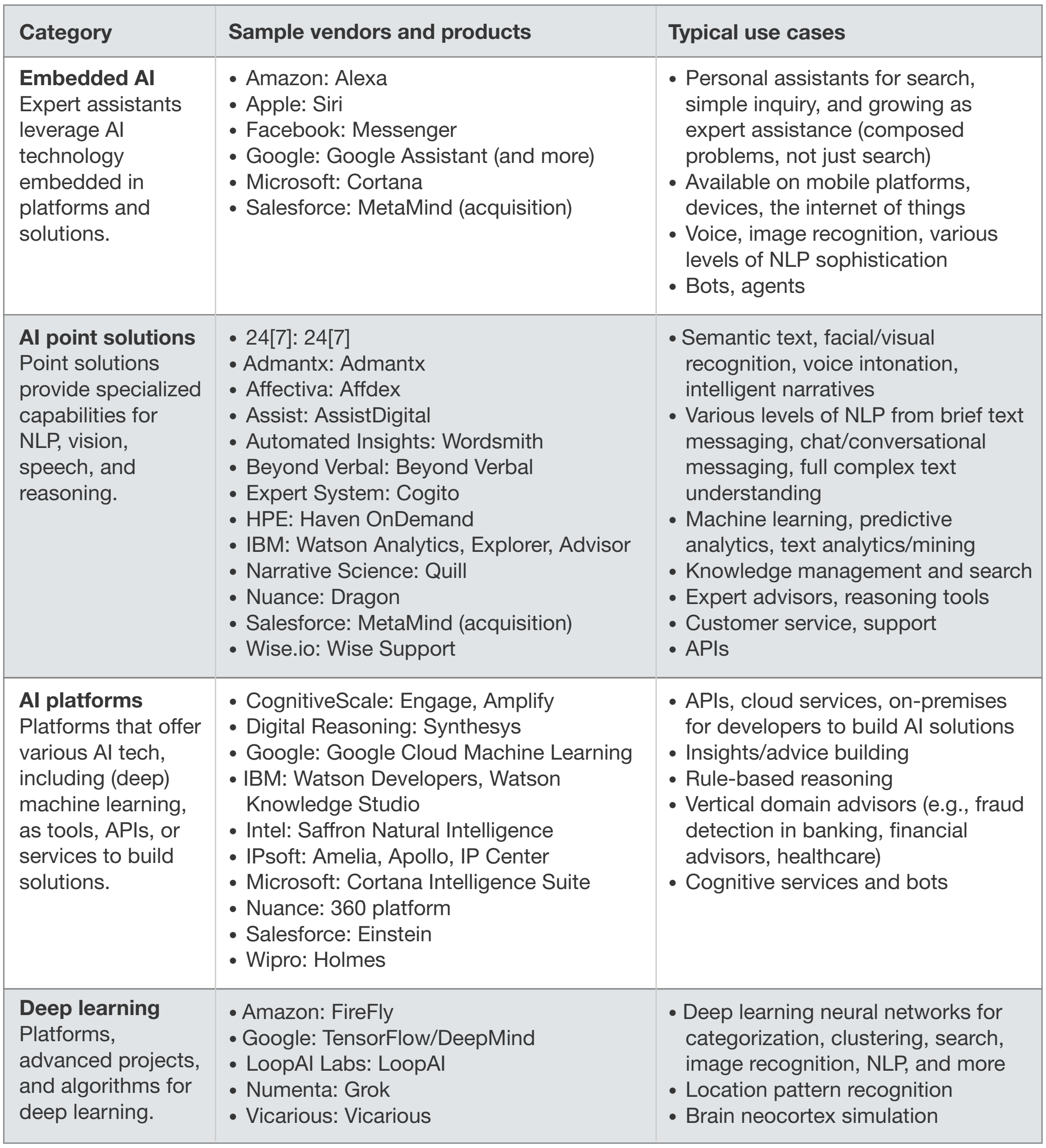
Introduction

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 [97] 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. Most prominently spearheading this wave of AI-first thinking is Google, as evident through their 2018 rebranding of *Google Research* to *Google AI* [69].

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 [55] to access the cloud service.

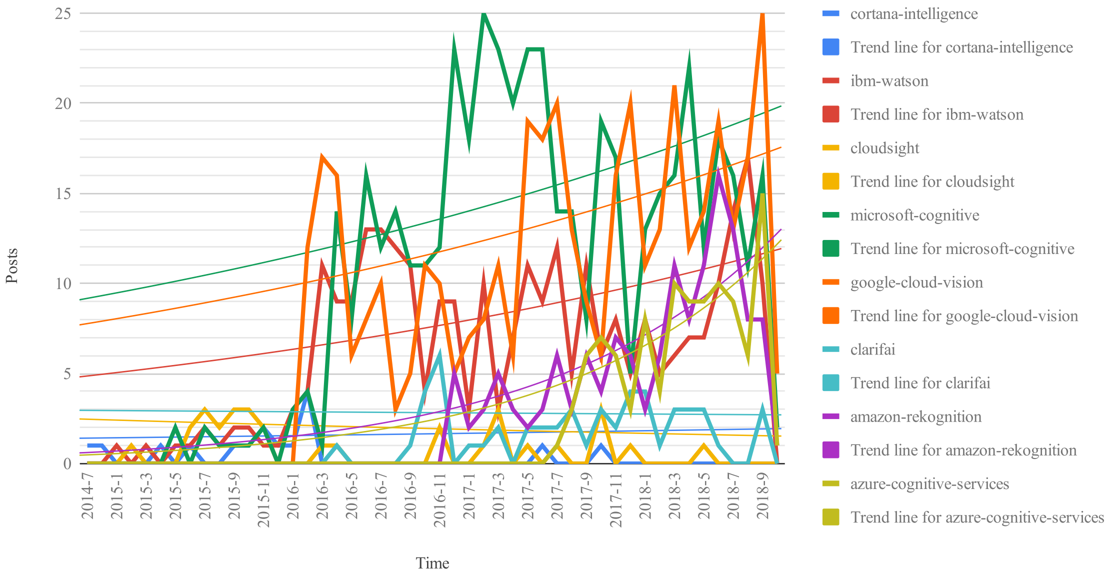
To contrast this with more traditional means, a developer may choose to write up a deep-learning NN (for example) and train it using their own dataset. While this is laborious in time and demands significant knowledge in machine learning, the developer has full control over the models she creates. Alternatively, she may choose to download a pre-trained model and ML framework, such as Tensorflow [16]; 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 [97].)



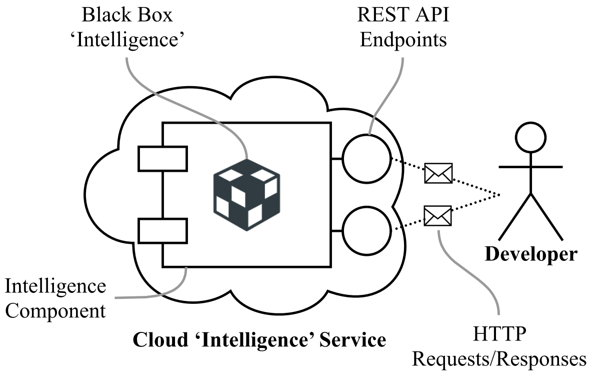
Category	Sample vendors and products	Typical use cases
Embedded AI Expert assistants leverage AI technology embedded in platforms and solutions.	• Amazon: Alexa • Apple: Siri • Facebook: Messenger • Google: Google Assistant (and more) • Microsoft: Cortana • Salesforce: MetaMind (acquisition)	• Personal 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 StackOverflow under popular computer vision cloud intelligence services.



With less time and skill required to build AI-first apps using these cloud services, these services have begun to gain traction within developer circles: Figure 1.2 shows the increasing trend of posts since 2014 on StackOverflow that categorise popular computer vision cloud APIs.[1] A growing popularity into such 'off-the-shelf' cloud services sparked varied nomenclature: *Cognitive Applications* and *Machine Learning Services* [72] or *Artificial Intelligence/Machine Learning as a Service* [122] are some coined phrases in which the intelligence is provided as a service via the cloud. We henceforth refer to such services under the broader term 'Cloud Intelligence Services' (CISs), and diagrammatically express their usage within Figure 1.3.

Figure 1.3: Overview of Cloud Intelligence Services.



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

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 [12] or ScaleAPI [14]. 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 computer vision CIS analysers [8, 3, 1, 13, 9, 5, 4, 7, 10, 15, 11, 6, 2]. The ubiquity of computer vision CISs is exemplified through evermore growing applications that use these APIs: aiding the vision-impaired [120, 43], accounting [98], data analytics [79], and student education [46]. 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 [19, 33] to describe “systems whose internal mechanisms are not fully open to inspection” [19].

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 [112] against its training dataset. As an example, a computer vision CIS 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 [115, 125] written by providers, while providers should consider mismatches between the developer's conceptual knowledge of the API its implementation [90]. 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 [26] 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 computer vision CISs. We explore more of these facets in the motivating scenario below.

1.1.2 Motivating Scenario

How do developers work with a CIS? How usable are these APIs, and how well do developers understand the non-deterministic and stochastic nature of a deep-learning cloud-based API? To motivate such a scenario, let us introduce a fictional software developer named Pam.

Pam wants to develop a social media photo-sharing mobile app that analyses her and her friends photos. Pam 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 her friends as well as common objects (e.g., all photos with her Border Collie dog, all photos taken on a beach on a sunny day).

Rather than building a computer vision engine from scratch, which would take far too much time and effort, Pam thinks she can achieve this using one of the common computer vision CISs. Pam comes from a typical software engineering background and has insufficient knowledge of key computer vision terminology and no understanding of the processes behind deep-learning. She ultimately believes all are APIs alike and internalises a deterministic mindset of them; when she decides on one of the three APIs, she expects a static result always. As she expects the same for whenever she calls, for example, any substring API with the call (or similar) of `substring("doggy", 0, 2)` and would expect the response ‘`dog`’ as its output.

To make an assessment of these APIs, she tries her best to read through the documentation of some computer vision APIs, but she has no guiding framework to help her choose the right one. Some of the questions that may come to mind include:

- What does confidence mean? Aren’t these APIs consistent?
- Will she need a combination of many computer vision APIs to solve this task?
- How does she know when there is a defect in the response? How can she report it?
- How does she know what labels the API can pick up, and what labels it can’t?
- How does she know when the models update? What is the release cycle?

- How does it describe her photos and detect the faces?
- How can she interpret the results if she disagrees with it to help improve her app?

Dazzled by this, she does some brief reading on Wikipedia but is confused by the immense technical detail to take in. She would like some form of guiding framework to assist her and in software engineering terms she can understand.

1.2 Research Goals

In this thesis, we explore the effect stochastic and probabilistic systems play on the usability of APIs with respect to computer vision CISs. Our 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 operationalisation such systems.

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. We propose three major bodies of work below.

Goal 1: ***Understand developers' mindsets towards computer vision CISs.***

Goal 2: ***Explore quality factors and assurances required for building computer vision CISs.***

Goal 3: ***Provide an evaluation framework developers wishing to use computer vision CISs.***

Chiefly, we can specify the following high-level research questions:

RQ1. How do software engineers understand and evaluate computer vision CISs for use in both generic and specific applications?

RQ2. Do software engineers follow best practices when evaluating computer vision CISs? How does this compare to actual practice?

RQ3. What is needed to improve the state-of-the-art of computer vision CISs in terms of API documentation?

RQ4. What aspects of validation and verification can be improved in the field of computer vision CISs?

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 computer vision CISs space will be generalisable to other areas, such as time-series information, natural language processing and others.

1.3 Methodology

For this study, we propose running several experiments involving developers and several computer vision CISs, using action-based mixed method approaches and involving documentary analysis. This study will organically evolve by observing phenomena surrounding computer vision API internal quality, chiefly their documentation and responses. We adopt a mixed methods approach, performing both qualitative and quantitative data collection on these two key aspects by using documentary research methods for inspecting the API documentation and structured observations to quantitatively analyse the results over time (RQs 3 and 4).

Our first proposal for usability studies will survey a number of developers from various levels of seniority and experience (gathering such demographical data to assess a wider sample size) to provide insight into how these developers perceive the non-deterministic nature of computer vision APIs, asking them specific questions about their conceptual understanding of computer vision to identify any outstanding gaps in their knowledge and factor this into known literature (RQs 1 and 2).

We will then conduct a structured interview with a 'mock' computer vision API to remove any developer bias toward any one particular computer vision API that already exists and by which the developer may have already used in the past. Here, we will investigate if developers have any patterns of practice and if they conform to software engineering best practices (RQs 1, 2 and 3).

From these insights, we can then develop a series of assistive recommendations that aide in improving the validation and verification of the existing computer vision API tooling. This may involve a third party tool that helps developers evaluate which particular API is right for their specific computer vision use case.

Chapter 2

Literature Review

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 computer vision CISs (e.g., Google Cloud Vision [8], AWS Rekognition [1], Azure Computer Vision [3], Watson Visual Recognition [9] 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 impact does this mindset mismatch have on software quality in the applications built using a CIS? What assurances are needed for developers to know that this mindset mismatch exists, and what can we learn from common software engineering practices (e.g., [117, 136])? Throughout this chapter, we will review the core principles of this 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 [116]

The philosophical viewpoint of ‘quality’ remains highly debated and there are multiple facets to perceive this complex concept [62]. 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 [76]. 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” [75].

Using this sentence, what characteristics exist for non-deterministic systems like that of a computer vision CIS? 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’ [63, 42], ‘meeting or exceeding customer expectation’ [109], or ‘fitness for use’ [85]—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 [117] adapted the manufacturing-oriented view of quality from [24] 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 [113, 114], Pham recounts the tale of a high-school student who sat a standardised test send out to 350,0000 students [141]. 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 an poor question being written, and poor process standards to check if their ‘correct’ answers were indeed 100% correct. In the words of Boehm [28], 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 [27] 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 [117], data sourced from Cigital Inc. [39] 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 [142].

Formally, we refer to the IEEE Standard Glossary of Software Engineering Terminology [73] 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 2.1).

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 [136]. Though it is commonplace for providers to have a 'ship-first-fix-later' mentality of 'good-enough' software [145], 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 computer vision CIS, 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?

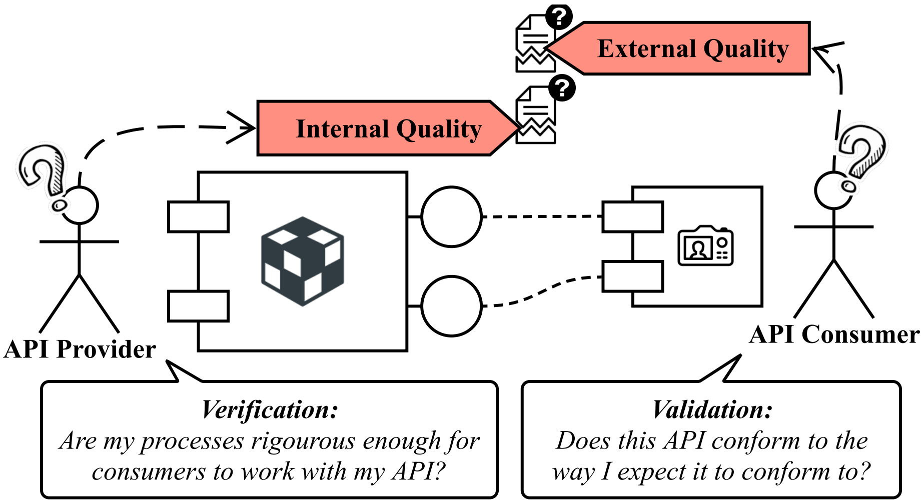

External Quality
Internal Quality
API Provider
API Consumer
Verification:
Are my processes rigourous enough for consumers to work with my API?
Validation:
Does this API conform to the way I expect it to conform to?

Figure 2.1: The 'leakage' of internal quality into the API consumer's product and external quality imposing on the API provider.

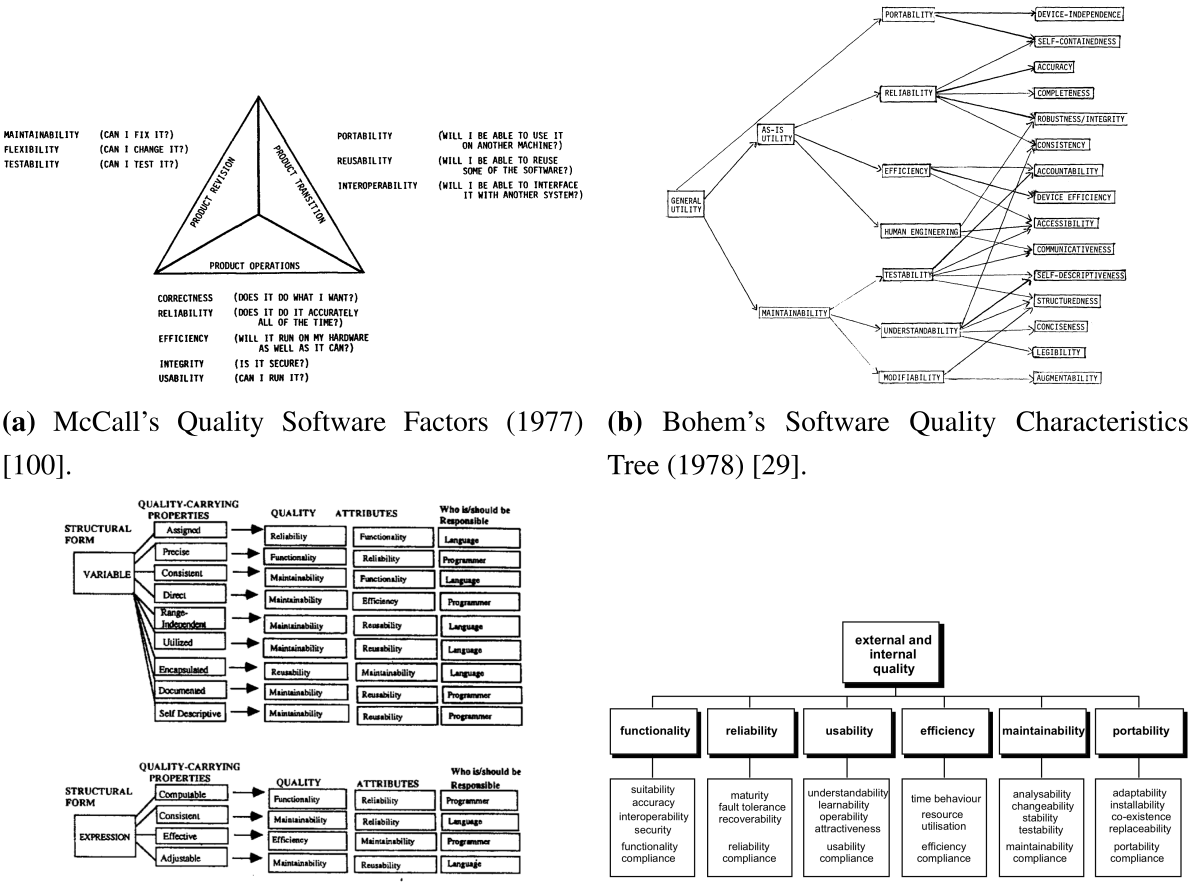
Therefore, just as the high-school student's test answers in the example we opened with are both correct and incorrect at the same time, so do CISs in returning a probabilistic result: no result is certain. While VV has been investigated in the area of mathematical and earth sciences for numerical probabilistic models and natural systems [107, 129], 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.

Quality attribute assessment models are an early concept in software engineering, and systematically evaluating software quality appears as early as 1968 [128]. This study introduced 'attributes' as a "prose expression of the particular quality of desired software" (as worded by Boehm et al. [29]) 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 [100, 37]. 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 2.2a). 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. [29] 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 2.2b. In the mid-1990s, Dromey's interpretation [52] defined a set of quality-carrying properties with structural forms associated to specific programming languages and conventions (Figure 2.2c). 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-9126 [77] (the Software Product Evaluation Characteristics, Figure 2.2d), which has since recently been revised to ISO-25010 with the introduction of the SQUARE model [74]. An extensive review on the development of quality models in software engineering is given in [17].


MAINTAINABILITY (CAN I FIX IT?)
FLEXIBILITY (CAN I CHANGE IT?)
TESTABILITY (CAN I TEST IT?)
PRODUCT REVISION
PRODUCT TRANSITION
PRODUCT OPERATIONS
PORTABILITY (WILL I BE ABLE TO USE IT ON ANOTHER MACHINE?)
REUSABILITY (WILL I BE ABLE TO REUSE SOME OF THE SOFTWARE?)
INTEROPERABILITY (WILL I BE ABLE TO INTERFACE IT WITH ANOTHER SYSTEM?)
CORRECTNESS (DOES IT DO WHAT I WANT?)
RELIABILITY (DOES IT DO IT ACCURATELY ALL OF THE TIME?)
EFFICIENCY (WILL IT RUN ON MY HARDWARE AS WELL AS IT CAN?)
INTEGRITY (IS IT SECURE?)
USABILITY (CAN I RUN IT?)
GENERAL UTILITY
AS-IS UTILITY
MAINTAINABILITY
PORTABILITY
RELIABILITY
EFFICIENCY
HUMAN ENGINEERING
TESTABILITY
UNDERSTANDABILITY
MODIFIABILITY
DEVICE-INDEPENDENCE
SELF-CONTAINEDNESS
ACCURACY
COMPLETENESS
ROBUSTNESS/INTEGRITY
CONSISTENCY
ACCOUNTABILITY
DEVICE EFFICIENCY
ACCESSIBILITY
COMMUNICATIVENESS
SELF-DESCRIPTIVENESS
STRUCTUREDNESS
CONCISENESS
LEGIBILITY
AUGMENTABILITY

(a) McCall's Quality Software Factors (1977) [100].

(b) Bohem's Software Quality Characteristics Tree (1978) [29].

STRUCTURAL FORM
QUALITY-CARRYING PROPERTIES
QUALITY ATTRIBUTES
Who is/should be Responsible
VARIABLE
Assigned Reliability Functionality Language
Precise Functionality Reliability Programmer
Consistent Maintainability Functionality Language
Direct Maintainability Efficiency Programmer
Range-Independent Maintainability Reusability Language
Utilized Maintainability Reusability Language
Encapsulated Reusability Maintainability Language
Documented Maintainability Reusability Programmer
Self Descriptive Maintainability Reusability Programmer
STRUCTURAL FORM
QUALITY-CARRYING PROPERTIES
QUALITY ATTRIBUTES
Who is/should be Responsible
EXPRESSION
Computable Functionality Reliability Programmer
Consistent Maintainability Reliability Language
Effective Efficiency Maintainability Programmer
Adjustable Maintainability Reusability Language
external and internal quality
functionality
reliability
usability
efficiency
maintainability
portability
suitability
accuracy
interoperability
security
functionality compliance
maturity
fault tolerance
recoverability
reliability compliance
understandability
learnability
operability
attractiveness
usability compliance
time behaviour
resource utilisation
efficiency compliance
analysability
changeability
stability
testability
maintainability compliance
adaptability
installability
co-existence
replaceability
portability compliance

(c) Dromey's quality-carrying properties and programming languages (1995) [52].

(d) ISO software product evaluation characteristics (1999) [77].

Figure 2.2: The brief overview of the development of software quality models since 1977.

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 [101].

Boehm et al. Code possesses the characteristic *reliability* to the extent that it can be expected to perform its intended functions satisfactorily [29].

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 [51].

ISO-9126 The capability of the software product to maintain a specified level of performance when used under specified conditions [77].

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 computer vision CIS 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 The Importance of Model Interpretability

As the rise of applied AI increases, the need for engineering interpretability around models becomes paramount. Model interpretability has been stressed since early machine learning research in the late 1980s and 1990s (such as Quinlan [119] and Michie [104]), and although there has since been a significant body of work in the area [134, 21, 121, 34, 127, 96, 30, 84, 20, 61, 44, 144, 23, 54, 95, 99, 110, 146], it is evident that ‘accuracy’ or model ‘confidence’ is still used as a primary criterion for AI evaluation [70, 80, 135]. 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” [57], a problem commonly referred to as the *datashift* problem [138]. Analogously, Freitas [57] 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.*** *[57]*

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 [96]), what is known is (i) there exists a critical trade-off between accuracy and interpretability [56, 83, 87, 65, 48, 152], and (ii) a single quantifiable value cannot satisfy the subjective needs of end-users [57]. As ever-growing domains ML become widespread[1], these applications engage

[1] In areas such as medicine [22, 93, 111, 123, 153, 144, 84, 53, 151, 81, 34], bioinformatics [58, 140, 86, 47,

end-users for real-world goals, unlike the aims in early ML research where the aim was to get AI working in the first place. In safety-critical systems where AI provide informativeness to humans to make the final call (see [36, 88, 71]), 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" [106] to end-users. **Therefore, when human-decision makers must be interpretable themselves [124], 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. [50] discuss when explanations are not provided from a legal stance—for instance, those affected by algorithmic-based decisions have a 'right to explanation' [64, 147] under the European Union's GDPR[3]. But, explanations are not the only way to ensure AI accountability: theoretical guarantees (mathematical proofs) or statistical evidence can also serve as guarantees [50], however, in terms of explanations, what form they take and how they are proven correct are still open questions [96].

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' [102] 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 [133] applies, just as others have done with similar issues in the SE realm [105, 149] (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 [50] 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*

82], finance [21, 71, 45] and customer analytics [146, 95].

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

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

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 2.3.

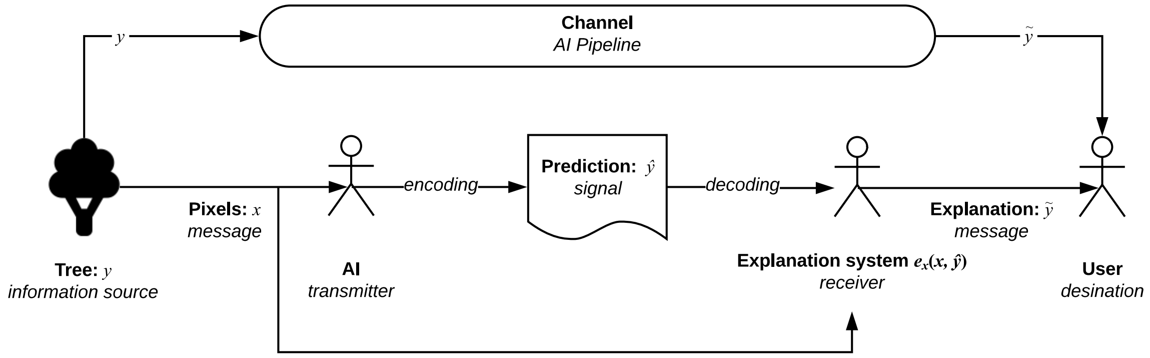


Figure 2.3: Theory of AI communication from information source, y, to intended user as explanations $\tilde{y}$.

2.2.3 Mechanics of Model Interpretation

How do we interpret models? Methods for developing interpretation models include: decision trees [32, 66, 41, 118, 126], decision tables [95**?**] and decision sets [92, 106]; input gradients, gradient vectors or sensitivity analysis [131, 121, 94, 127, 21]; exemplars [89, 59]; generalised additive models [36]; classification (*if-then*) rules [143, 31, 40, 108, 150] and falling rule lists [134]; nearest neighbours [99, 132, 139, 148**?**] and Naïve Bayes analysis [22, 93, 91, 153, 103, 60, 38, 68]. Several cross-domain studies have assessed the interpretability of these techniques against end-users, measuring response time, accuracy in model response and user confidence [71, 67, 18, 137, 130, 58, 99, 146], although it is generally agreed that decision rules and decision tables provide the most interpretation in non-linear models such as SVMs or NNs [58, 99, 146]. For an extensive survey of the benefits and fallbacks of these techniques, we refer to Freitas [57] and Doshi-Velez and Kim [49].

As it stands, AI presents an issue with. (For a detailed discussion, see Doshi-Velez et al. [50].

2.3 API Documentation and Standards

⟨ 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. [78] 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 [35] that supports all underlying layers [25]. Figure 2.4 captures these concepts in further detail.

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

2.5 Cognitive Biases

⟨ TODO: **Background; what are cognitive biases and how does it relate to SE?** ⟩

⟨ TODO: **Literature of CB specifically in SE.** ⟩

⟨ TODO: **List potential CBs with relation to the AI-based systems.** ⟩

2.6 UX Consistency Principle

⟨ TODO: **Background; what is UX consistency? What does it advocate for and why?** ⟩

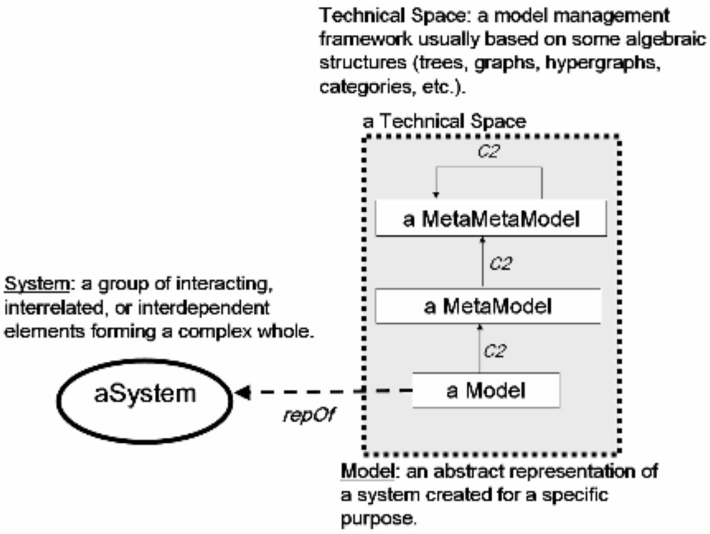

Technical Space: a model management framework usually based on some algebraic structures (trees, graphs, hypergraphs, categories, etc.).
a Technical Space
C2
a MetaMetaModel
C2
a MetaModel
C2
a Model
System: a group of interacting, interrelated, or interdependent elements forming a complex whole.
aSystem
repOf
Model: an abstract representation of a system created for a specific purpose.

Figure 2.4: Systems, models and technical spaces. (From [25].)

⟨ TODO: What lessons can we learn from UX consistency, and how can we apply it to SE? ⟩

⟨ TODO: What are the gaps in SE that do not conform to practices of UX consistency w.r.t. AI systems development? ⟩

Chapter 3

Methodology

3.1 Data Collection and Ethics

3.2 Approach

3.3 Evaluation Methods

3.4 Threats to Validity

3.4.1 Internal Threats

3.4.2 External Threats

3.4.3 Construct

Chapter 4

Project Status

4.1 Completed Work

4.2 Impact

4.3 Timeline

Chapter 5

Conclusion

References

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

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

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

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

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

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

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

[8] "Vision API - Image Content Analysis — Cloud Vision API — Google Cloud," https://cloud.google.com/vision/, accessed: 13 September 2018.

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

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

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

[12] "Amazon Mechanical Turk," https://www.mturk.com, accessed: 15 October 2018.

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

[14] "Scale: API for Training Data," https://www.scaleapi.com, accessed: 15 October 2018.

[15] "Image recognition - talkwalker," https://www.talkwalker.com/image-recognition, accessed: 13 September 2018.

[16] 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/

[17] 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.

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

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

[20] 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.

[21] 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.

[22] 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.

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

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

[25] 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.

[26] 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.

[27] 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.

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

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

[30] 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.

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

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

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

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

[35] 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.

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

[37] 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.

[38] 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.

[39] Cigital Inc., “Case study: Finding defects earlier yields enormous savings,” 2003.

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

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

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] P. Domingos, “Occam’s two razors: The sharp and the blunt,” in *KDD*, 1998, pp. 37–43.

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

[50] 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.

[51] R. G. Dromey, “A Model for Software Product Quality,” *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 146–162, Feb. 1995.

[52] ——, “A model for software product quality,” *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 146–162, 1995.

[53] 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.

[54] 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.

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

[56] 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.

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

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

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

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

[61] 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.

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

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

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

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

[66] 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.

[67] 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.

[68] 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.

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

[70] 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.

[71] 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.

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

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

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

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

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

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

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

[79] 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.

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

[81] 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.

[82] 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.

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

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

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

[86] 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.

[87] 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.

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

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

[90] 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.

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

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

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

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

[95] 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.

[96] Z. C. Lipton, “The Mythos of Model Interpretability.” *CoRR*, 2016.

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

[98] 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.

[99] 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.

[100] J. A. McCall, “Factors in software quality,” *US Rome Air development center reports*, 1977.

[101] 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.

[102] J. McCarthy, “Programs with Common Sense,” Cambridge, MA, USA, Tech. Rep., 1960.

[103] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, “Machine Learning and Statistical Classification of Artificial intelligence,” 1994.

[104] D. Michie, “Machine Learning in the Next Five Years.” *EWSL*, 1988.

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

[106] 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.

[107] 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.

[108] 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.

[109] 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.

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

[111] 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.

[112] J. Pearl, “The Seven Tools of Causal Inference with Reflections on Machine Learning,” 2018.

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

[114] ——, “System Software Reliability,” Springer London, London, 2006.

[115] 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.

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

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

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

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

[120] 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.

[121] 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.

[122] 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.

[123] 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.

[124] G. Ridgeway, D. Madigan, T. Richardson, and J. O’Kane, “Interpretable Boosted Naïve Bayes Classification.” *KDD*, 1998.

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

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

[127] 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.

[128] 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.

[129] 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.

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

[131] 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.

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

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

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

[135] 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.

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

[137] 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.

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

[139] 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.

[140] 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.

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

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

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

[144] 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.

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

[146] 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.

[147] 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.

[148] 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.

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

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

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

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

[153] 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.