Ways of Applying Artificial Intelligence in Software Engineering

Robert Feldt, Francisco G. de Oliveira Neto, and Richard Torkar
Chalmers and the University of Gothenburg
Gothenburg, Sweden
[robert.feldt|gomesf|torkarr]@chalmers.se

ABSTRACT

As Artificial Intelligence (AI) techniques have become more powerful and easier to use they are increasingly deployed as key components of modern software systems. While this enables new functionality and often allows better adaptation to user needs it also creates additional problems for software engineers and exposes companies to new risks. Some work has been done to better understand the interaction between Software Engineering and AI but we lack methods to classify ways of applying AI in software systems and to analyse and understand the risks this poses. Only by doing so can we devise tools and solutions to help mitigate them. This paper presents the AI in SE Application Levels (AI-SEAL) taxonomy that categorises applications according to their *point* of AI application, the type of AI *technology* used and the *automation level* allowed. We show the usefulness of this taxonomy by classifying 15 papers from previous editions of the RAISE workshop. Results show that the taxonomy allows classification of distinct AI applications and provides insights concerning the risks associated with them. We argue that this will be important for companies in deciding how to apply AI in their software applications and to create strategies for its use.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; • **Software and its engineering**;

KEYWORDS

Taxonomy, Software Engineering, Artificial Intelligence

ACM Reference Format:
Robert Feldt, Francisco G. de Oliveira Neto, and Richard Torkar. 2018. Ways of Applying Artificial Intelligence in Software Engineering. In *Proceedings of 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE'18).* ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

Artificial Intelligence (AI) has shown a lot of promise in the last decades but with the recent resurgence of interest and improved results on real-world tasks the field is undergoing explosive growth. Many of the improved results have come from larger and more complex neural networks, stacked many layers deep (for so called Deep Learning), but much of progress can also be attributed to larger data sets and large-scale learning/training on GPUs [40]. But the renewed interest and increasing amount of resources has also lead to breakthroughs in related AI technologies, e.g. Bayesian statistics [7, 21], generative models [22], and probabilistic programming/induction [29].

However, there has been recent criticism that many of these approaches to building more intelligent software are too far from human-level intelligence and, thus, are not likely to be enough [30, 33]. Instead the critics argue that we actually need algorithms that build and extend causal models, can learn from very few examples (one- or few-shot learning), and can reason symbolically with the patterns and knowledge they extract from sensors [30, 33].

Regardless if the current set of AI technologies will be enough to reach human-level intelligence or not it is clear that software systems will increasingly incorporate them as components and sub-systems. The form of the solutions produced from these AI/ML technologies often look inherently different from the software that is normally developed and deployed. Thus, not only does the AI technology itself change quickly and at an increasing pace, the solutions it provides typically look very different from what software organisations and engineers are used to. This poses a new and unique set of risks and opportunities for software organisations and they need to understand and analyse these risks to select appropriate strategies.

This hybridisation of AI/ML and software engineering is inevitable also in another sense. There will be ample opportunity to apply AI and ML models to improve software development itself. Software engineers are close to these technologies and are likely to be early adopters in applying them on their own problems, methods and tools. This is also helped by the trend that AI/ML technologies is increasingly componentised and can be more easily used and reused, even by non-experts. Advances in software engineering allow AI technologies to be packaged and easily reusable through RESTful APIs[1] as automated cloud solutions, which can use multiple technologies before selecting and automatically tuning one/several[2]. As AI becomes more accessible its use can be expected to increase even more.

Given the current expansion of the field and the large number of different ways that AI/ML can be applied during the engineering of software and in the software systems themselves, there is a risk of confusion and miscommunication. If we do not have shared terms to describe the terrain and an overview of possibilities it becomes harder for engineers and software organisations to properly assess risks and discuss mitigation strategies. In our experience, of working with and applying AI/ML technologies to software engineering problems in industry, we have seen this first-hand [1, 6, 11, 16–19].

[1] https://bigml.com/api
[2] https://cloud.google.com/automl

RAISE'18, May 2018, Gothenburg, Sweden

Clear definition of terms and a way to classify and understand opportunities and risks can be critical in improving communication and enabling more detailed weighing of alternatives. This forms the basis of creating organisational strategies. Here, we propose an initial taxonomy with which to analyse and understand different ways of applying artificial intelligence and machine learning in software engineering. To show the utility of the approach we apply it to a sample of papers that have been published at the RAISE workshop on realizing synergies between AI and software engineering.

2 BACKGROUND AND RELATED WORK

Since the early days when Barr and Feigenbaum [4] discussed the possibilities of combining AI with software engineering, there have been some attempts to classify the field to systematically be able to attack key challenges. In 1987, Barstow [5] presented a review of how one should apply AI techniques to software engineering problems. He distinguished between programming-in-the-small (by individuals or very small groups) and programming-in-the-large (by very large groups of people). He divided AI usage into five broad categories: Software engineering methodologies, programming techniques, the architecture of the target machine, the application domain, and the history of the target software.

At RAISE'12, Clifton et al. [9] provided an overview of machine learning and software engineering in health informatics by presenting ongoing work from several projects. Many of the presented cases show that the scale of clinical practice requires new engineering approaches from both disciplines. This is but one example where one sees that both AI and SE need new engineering approaches for specific domains. However, we also see a need to address challenges on a higher level of abstraction as Harman [24] pointed out in his RAISE keynote that same year. From Harman's perspective, the abstraction would serve us in developing "strategies for finding solutions rather than the solutions themselves." In this particular case, we are not looking specifically at finding strategies by the use of abstraction, but rather to provide researchers and practitioners with a view of AI in SE, and how 'embedded' we would wish for AI techniques to be, i.e., answering the why and what, while taking risk into consideration.

On that note, Davis et al. [10], propose a taxonomy of AI approaches for adaptive distributed real-time embedded (DRE) systems. The taxonomy classifies AI approaches according to five properties needed for adaptive DRE systems: *i*) supporting a distributed environment, *ii*) supporting real-time requirements, *iii*) supporting an embedded environment, *iv*) robustly handling new data, and *v*) incorporating new data into the approach as it becomes available while the system is running. Their classification is fine-grained and is limited to the context of DRE system, since the goal is identifying suitable AI applications.

Another example of a taxonomy tailored to specific AI applications in SE is proposed by Charte et al. [8]. Their taxonomy provides a broad view of autoencoders (AE) which are Artificial Neural Networks (ANNs) that produce codifications for input data and are trained so that their decodifications resemble the inputs as closely as possible. The AEs are used in different applications related to SE such as data compression, hashing and visualisation. The taxonomy is based on the different features of an AE, such as lower dimensionality, or noise tolerance. However, Charte et al. do not include AI technologies beyond AEs.

We could not find other taxonomies that covers classification within both areas of AI and SE; existing proposals focus on one or the other and are rather fine-grained. Unlike the existing taxonomies, the one we present in this paper, AI-SEAL (Artificial Intelligence in Software Engineering Application Levels), aims to be more general and allow classification beyond a specific subject matter, such as for a specific type of software system, e.g., DRE systems, or a specific type of AI technology, e.g., AEs. It has three main facets (dimensions) to explore different perspectives of AI in SE on a more general level, thus allowing us to cover more AI approaches and be more inclusive of other application domains within software engineering. By being more coarse-grained it can also be complemented by more detailed taxonomies, such as the ones described above, or additional, lower-level facets, as outlined at the end of the next section.

AI-SEAL also differentiates itself by targeting a different use case. While the taxonomies and categorisations we have described above focus on providing an overview of the field or outlining possibilities when applying AI technologies in SE, our overall aim is to support companies and organisations developing strategies for such applications. The main aim is thus to be able to estimate and analyse the risks and costs involved, not only the possibilities. When a certain type of technology is new there is a tendency to focus on possibilities and the upside of its application while real-world adoption requires a deeper understanding also of the downsides. Our focus is on helping organisations better understand the risks.

2.1 Methodology

In order to propose AI-SEAL we investigated the creation and usage of taxonomies in SE. There are several taxonomies proposed for SE in all of its different knowledge areas (e.g., requirements and testing [43]), but very few are created systematically [44]. Usman et al. performed a systematic mapping on the use of taxonomies in software engineering, and proposed a method to develop such taxonomies [44], which we used when defining AI-SEAL.

There are four phases in the Usman et al. method: *i*) planning, *ii*) identification and extraction, *iii*) design and construction, and *iv*) validation. During planning we decided to be inclusive of all knowledge areas within software engineering (e.g., testing, requirements, processes) and rather seek a more fundamental aspect of how AI is applied. We argue that this is natural since it would not matter much to the risks involved whether one applies AI to for example requirements or design; we argue that the main risks arise if the actual software itself changes shape and to what degree the engineers or, even later, the users, can change the proposals and/or decisions put forward or implemented by the AI or software created by it.

The goal with the identification and extraction phase is to identify the main categories and associated terms used in the taxonomy. The challenge was to identify terms and categories pertaining to the different SE knowledge areas and the variety of available AI technologies, not to mention that both fields are constantly changing and evolving. Based on our own experience from applying AI in SE, and from reading relevant papers published in the last years,

we identified many key facets/dimensions but then filtered them down.

We extracted three key facets for our taxonomy: point of application, automation level and AI technology. We decided to use a faceted analysis because this type of classification structure is suitable for new and evolving fields, since complete knowledge related to the subject area is not required or available [44].

The design and construction phase presents the different levels within those facets, how they were chosen and to what extent they are connected to the taxonomy's purpose and usage (we explain the details in Section 3). Lastly, the validation phase aims at demonstrating how the subject matter can be classified using the proposed taxonomy. Literature provides three distinct validation methods: Orthogonality demonstration, benchmarking and utility demonstration [44]. We chose the latter and classified a set of AI applications reported in previous editions of the Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE). Additionally, utility demonstration is the most reported method to validate taxonomies in SE, and it allows us to showcase the classification extent of the taxonomy [44].

3 TAXONOMY FOR AI-IN-SE APPLICATION LEVELS

3.1 Overview and Description

The purpose with the Artificial Intelligence in Software Engineering Application Levels (AI-SEAL) taxonomy is to support researchers and practitioners to communicate, understand and discuss the pros and cons of applying AI approaches when developing and in running software systems. We argue that since the end goal of any software engineering process is to deliver a running software system we cannot exclude the actual use of AI during system execution.

An explicit goal was to keep the number of facets to a minimum; in the end we propose only three. Even though they can be further sub-divided and additional facets can help detail them more we argue that this can be left for future work. A simpler taxonomy is more likely to be useful, in particular for practitioners and companies. The three facets we propose as critical are Point of Application (PA), Type of AI (TAI) applied, and Level of Automation (LA) offered.

The *point of application* (PA) includes both the 'when' (in time) and the 'on what' (location) the AI technology is being applied (Figure 1). There are three major levels of this facet, two that are relevant before deployment of the software system (*process* and *product*) while the third is post-deployment representing the *runtime* application of AI in a software system.

The *process* level indicates that the AI is applied in the software development process and does not necessarily affect, directly, the source code that will be deployed. An example would be test analytics, which could be used to optimise testing, but it does not by itself directly alter the code, e.g., [2, 18]. In contrast, the *product* level indicates that the AI directly affects the source code. A concrete example would be automated program repair, which manipulates the code directly to automatically fix defects [35].

The *runtime* level represents AI applications that affect the deployed software system during runtime. The canonical example would be autonomous and self-adaptive software systems in which

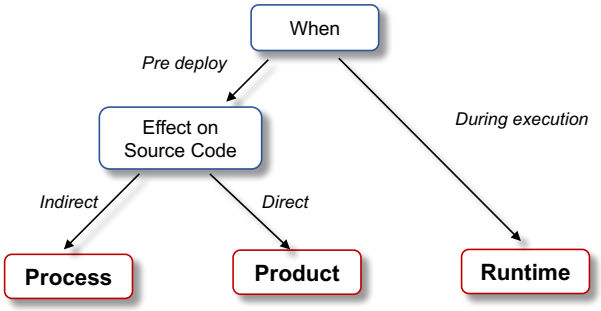

Figure 1: Overview of the different points of application (PA) in the AI-SEAL taxonomy.

some AI technology is learning and changing the system itself in a feedback loop [42]. A more mundane, but recent example, would be the online learning of more optimal data structures and database indices based on the actual data stored during operation, in line with recent results from Google [28].

Some applications can span these main levels of PA or can be viewed as borderline between levels. An example would be autonomous driving software that includes an ANN. In a situation where the AI develops a part of the code, which is then compiled into the binary that goes into the final product, it would be classified under the *product* application level. However, if the ANN is dynamically updating itself using runtime information from the executing software, then its PA classification would be set to the *runtime* application of AI. For risk analysis we argue it makes more sense to then select the latter (higher) level, i.e., runtime over product and product over process. The reason is that the higher the level the less time there is, in general, for humans to intervene or even to analyse the result of the applied AI technology.

The next facet of AI-SEAL covers the *Type of AI* (TAI) that is applied. Since there is not even a consensus around what AI is it becomes hard to propose a particular and stable set of levels for this facet. This facet is, thus, most likely to need to change as progress is made and new types of AI approaches are proposed. As a starting point we propose that 'the five tribes of AI' classification introduced by Domingos [15] can be useful:

- Symbolist, e.g., inverse deduction.
- Connectionist, e.g., backpropagation.
- Evolutionaries, e.g., genetic programming.
- Bayesians, e.g., probabilistic inference.
- Analogizers, e.g., kernel machines.

Even though these five tribes capture general types of AI technology it is clear that the TAI facet can be made more detailed and divided into further sub-dimensions, depending on the representations, algorithms, and artefacts used in a particular application of AI. However, we argue that specifying sub-dimensions hinders the practical use of the taxonomy since it can be confusing to precisely distinguish among the different existing algorithms, mainly if more than one AI technique is involved in, e.g., the product. In other words, AI-SEAL users can choose to go deeper into that facet within their domain-specific constraints, but we do not incorporate those sub-dimensions into the taxonomy itself. The main purpose of

Table 1: Levels of automation (LA) of decision and action selection (from [20] and [41])

10.	Computer makes and implements decision if it feels it should, and informs human only if it feels this is warranted.
9.	Computer makes and implements decision, and informs human only if it feels this is warranted.
8.	Computer makes and implements decision, and informs human only if asked to.
7.	Computer makes and implements decision, but must inform human after the fact.
6.	Computer makes decision but gives human option to veto before implementation.
5.	Computer offers a restricted set of alternatives and suggests one, which it will implement if human approve.
4.	Computer offers a restricted set of alternatives and suggests one, but human still makes and implements final decision.
3.	Computer offers a restricted set of alternatives, and human decides which to implement.
2.	Computer offers a set of alternatives which human may ignore in making decision.
1.	Human considers alternatives, makes and implements decision.

the TAI facet is to consider the particular properties of the applied AI technology and how they interact with the PA and LA facets in a particular application. For example, in a product application the risk might be much higher with using a Connectionist AI technology, which produces an opaque neural net that is hard to analyse and test, than if using an evolutionary search process to find decision rules that are short and can be analysed before they are deployed.

The last facet is *the level of automation*, i.e., LA, which the AI application aims at or achieves. We base the levels of this facet on the Sheridan-Verplanck 10 levels of automation, an existing taxonomy from Automation/HCI research that focuses on human-computer decision making [20, 41] (Table 1).

The Sheridan-Verplanck taxonomy conveys how different human operators (e.g., a developer, tester, user, or any stakeholder in the software system) and the technical system (e.g., AI technology) should cooperate by sharing the control of determining and selecting options to implement tasks. At lower levels of automation, the AI technology simply provides data through, for instance, dashboards with descriptive and visual information, while the stakeholder responsible for understanding the information and determining the next course of action of the software.

As we climb up these levels, we allow the AI technology to be more autonomous by either allowing it to suggest alternatives to the human operator (Level 2) or even, at the top level, implement the decisions itself and only inform the human if it so decides (Level 10). Hence, the higher the level of automation, the more autonomous the AI technology becomes in making decisions related to the element, e.g., product, process, or runtime, where it is being applied. And, we argue, the higher the risk involved [3].

[3]We admit that there is also a case to be made that ultimately, and especially for some tasks, solutions based on AI might be less risky than having humans have the final say before taking action. However, we argue that the decision to trust AI solutions will have to be based on other risk-reduction strategies such as taking a general decision to trust certain types of systems based on empirical evidence of their safety. We thus see this as a future extension possibility rather than a threat to the taxonomy presented here.

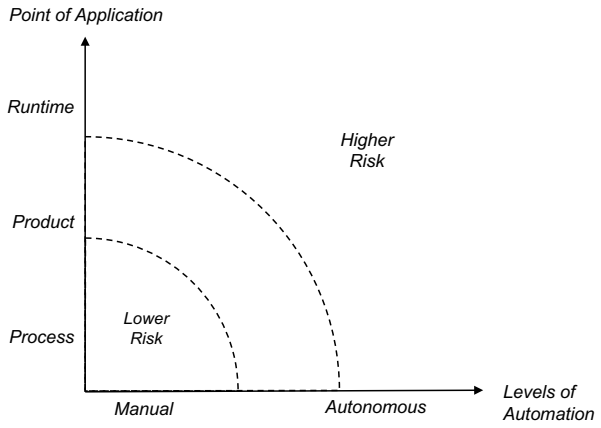

Figure 2: Levels of risk on the AI-SEAL taxonomy with respect to Points of Application (PA) and Levels of Automation (LA).

Note that one of the main benefits of the three facets of the taxonomy is to allow practitioners and researchers to classify and understand the risks involved in the AI application. If we consider the PA facet, we argue that the risk of applying AI increases as we move from *process* to the *runtime* level. At each step, there is more at stake since negative consequences due to the application of inappropriate AI technologies are worse/costlier to reverse closer to (or after) deploying the product. As such, PA is closely related to the size of the impact as well as level of control that developers have on the AI application. Similarly, the LA facet is riskier at higher levels, since stakeholders will have less time to reverse decisions when the AI has a higher degree of autonomy (see the risk boundaries in Figure 2).

Thus, if an AI technology is new to a company, practitioners should start at low levels of automation (LA) to allow more human intervention, as well as at a 'lower' point of application (PA), where potential issues with introducing AI will not be directly introduced to the source code. Then, by building more experience one can expand to explore higher levels of automation and points of application of the AI technology in SE and software systems.

4 EVALUATION

We evaluate the AI-SEAL taxonomy through a utility demonstration [44] by classifying papers from previous editions of the RAISE workshop. The workshop focuses on papers showcasing applications of AI in SE, in a broad sense, and is, thus, a suitable venue for identifying relevant papers.

The first instance of RAISE was held in 2012 and, except for 2017, it has been running annually (2012–2016). A total of 44 papers have been presented at RAISE over the years. Based on the title we excluded ten papers that we assessed as not presenting a specific application/solution of AI in SE, e.g., surveys, or papers presenting challenges or open issues. For the remaining set of papers we then included the most recent ones, published at RAISE 2015 (six out of

Table 2: Classification of 15 RAISE papers according to the AI-SEAL taxonomy. Papers #6 and #15 both discuss Runtime and higher levels of automation of the AI so we mark them as borderline FW (Future Work) below and list them twice.

ID	Reference	Appl. Point (PA)	Type of AI (TAI)	Level of Auto (LA)
#1	[26]	Process	Analogizer	2
#2	[27]	Process	Analogizer	4
#31	[32]	Process	Connectionist	7
#34	[38]	Process	Symbolist	9
#36	[39]	Process	Analogizer	2–3
#37	[37]	Process	Symbolist	2–3
#38	[23]	Process	Symbolist	2
#40	[13]	Process	Evolutionary	2–3
#41	[14]	Process	Analogizer	2–3
#42	[36]	Process	Analogizer	2–3
#43	[3]	Process	Analogizer	2–3
#44	[34]	Process	Analogizer	2–3
#6	[12]	Product	Symbolist	4
#15	[25]	Product	Analogizer	7
#35	[31]	Product	Symbolist	9
#6	[12]	$(\text{Runtime})_{\text{FW}}$	Symbolist	$(8)_{\text{FW}}$
#15	[25]	$(\text{Runtime})_{\text{FW}}$	Analogizer	7

seven papers; one not found in the IEEE database) and 2016 (four papers), as well as a random sample of another five for a total of 15 papers[4]. We then applied the AI-SEAL taxonomy to these 15 papers. In Table 2 we show an overview of the results and below we describe four papers in more detail, to show the value of our approach.

Paper #1, written by Iliev et al. [26] uses an ontology, based on design information provided by stakeholders, to automatically predict the severity level of a defect. The AI suggests the severity levels to stakeholders who, in turn, can accept or ignore the suggestions. Therefore, the AI-SEAL classification of Paper #1, for PA, TAI and LA is, respectively, *Process*, *Level 2 of Automation* and included in the *Analogizer* tribe since the design information and the classification rules used by the ontology are defined in advance by stakeholders.

Paper #6, written by de Souza Alcantara et al. [12] presents an approach where a tool learns a set of gestures that UI designers can use to design gesture-based applications for multi-touch devices. We classify this as a *Symbolist* AI technology since it analyses the relations between and reasons about the different steps of a specific gesture, which are inferred based on a set of primitives. Hence, the designer first trains the tool to learn a set of gestures that can be used when designing the UI of the application. Then, during the actual design of the UI, IGT can detect gestures outside the standards and prompt the designer to ask whether the drawn gesture was a mistake. Authors state that the gestures definitions are not updated; therefore, the levels for PA and LA are, respectively, *Product* (the final gesture identification output from the tool in included in the developed application) and *Level 4*. However, authors

[4]https://goo.gl/ERPoUk

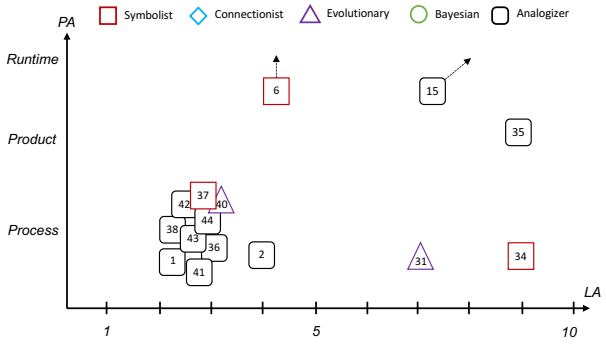

Figure 3: AI-SEAL classification of papers from different editions of RAISE. The papers used in this classification were [3, 12–14, 23, 25–27, 31, 32, 34, 36–39].

discuss future work where the interaction between designer and the tool is higher, and the gestures definitions are updated; this could move the technology to higher levels of LA and, possibly, PA.

Paper #15, written by Heitmeyer et al. [25], is titled *High assurance human-centric decision systems* and proposes an approach where AI techniques are used to detect and assist operators of a decision system that start to feel overloaded given the complexity of tasks in the decision system. They use two AI techniques, both from the *Analogizers* tribe, to predict human overload based on the past interactions of the operator with the system. Ultimately, the AI application should take over the system operation while alerting the operator ($LA = 7$). Therefore, the application is at the *product* level, since the overload model does not update during runtime, even though authors plan to address this issue in future work. This is an example how the application can actually begin at lower risk levels, and then evolve to higher risk areas of the AI-SEAL taxonomy.

Paper #31, written by Langer and Oswald [32], is titled *A self-learning approach for validation of communication in embedded systems*. The authors use neural networks (*Connectionist*) to learn which communication traces are valid in a distributed embedded system. The proposed approach is used for automated integration testing focusing the communication between the distributed components, thus being applied at the *process* level. Then, the AI automatically analyses the communication traces, and notify users only if an invalid communication trace is found ($LA = 7$).

We summarise the classification of the papers in our taxonomy in Figure 3 and Table 2. Most of the papers (53%) use analogizers, and, regarding the PA facet, most of them are applied at the process level (80%), and at lower levels of automation. In contrast, none of the evaluated papers proposed applications of AI technologies from the Bayesian tribe, and only two of them discussed Runtime application of the AI as future work. We argue that this more might show a bias of the RAISE workshop than anything fundamental; for example we gave two papers as examples of Runtime applications when introducing the facet. An explanation for the little use of Bayesian solutions might be that they are not as clearly seen as part of AI technology as part of advanced statistics; however, as Domingos makes clear in his book their view of AI is also valid and general [15].

We note that this is an initial proposal on how the taxonomy can be used to raise awareness of current state of art regarding AI applications in SE. Certainly, some applications will be more challenging than others to classify, particularly when the application or technology span across different levels of one or more facets or as new AI technologies and solutions are introduced.

5 DISCUSSION

We have introduced a taxonomy (AI-SEAL) having three facets that can be used to classify and analyse ways of applying AI in software engineering and thus help understand the associated risks and opportunities. Our main argument is that the risk one takes in applying AI relates to the level of control and the time given to exert control that the developers and users have (over) the decisions proposed or taken by the AI component. Together this explains two of the three facets; the third is used for basic characterisation of the AI technology being applied.

Our evaluation shows that AI-SEAL can classify applications regardless of the SE knowledge area (e.g., design, development, maintenance) and even domain-specific information (e.g., UI design, safety-critical systems, or object-oriented programming) involved. This is by design; we argue that a taxonomy needs to have this property to be generally useful. It can of course be complemented by additional facets or existing taxonomies to further detail the two main facets.

Similarly, the PA facet is inclusive of different application domains and types in SE, since it depends on elements present in any SE application, i.e., the source code of the system, that is eventually deployed, and a process to develop it. There are plenty of ways to expand on the PA facet, for example by using SWEBOK's knowledge areas to further describe what the AI is applied to and when. In fact, this possibility to update and expand the facets independently is one of the main benefits of a faceted taxonomy [44], thus being a suitable design choice for a taxonomy in dynamic fields such as AI and SE.

The demonstration (Figure 3) also reflects the risks of the classified applications. For instance, Paper #15 is an example of a software system controlled by an AI and a human operator that cooperate to control unmanned air vehicles (UAV), an example of autonomous vehicle, which are widely used by the military for surveillance and targeting [25]. Therefore, the risk related to that AI in SE application is high since mistakes caused by the AI could have adverse outcomes.

On the other hand, Paper #1 [26] has lower risk, where the AI automatically classifies the level of severity of found defects. Note that at $LA = 2$ the AI provides input, but stakeholders are the ones responsible to decide, so the development process can recover from eventual adversities with poor classification. Certainly, at higher levels of automation (e.g., LA ≥ 6), the risks would be higher since the AI becomes more autonomous and can disrupt the development process or lead to decisions that ultimately decrease quality.

On that note, the risks in Papers #2 and #6 differ mainly due to the PA levels. Paper #2 detects clones on source code during maintenance, thus having an impact on maintenance costs and internal quality (e.g., refactoring) as opposed to paper #6 that is used to assist its end-user (UI designers). Any mistakes in the AI application in Paper #6 affects the product, thus its impact on external quality is higher.

Certainly, the risk analysis is sensitive to other factors involving both the AI and SE parts. For example, the familiarity that the company and the engineers have with the particular AI technology being used will be important. However, the value of AI-SEAL is to give an overview of the risk and the possible impacts[5]. Therefore, we recommend practitioners interested in exploring possible AI application in their SE projects to begin in the lower risk areas, and then move towards riskier areas as they gather experience on the AI application in SE. For researchers, AI-SEAL is useful to map the field and identify areas to employ more research effort.

In it's current form AI-SEAL does not help in specific classification of AI technology, for example helping practitioners to understand the distinction between different ANN models, or exposing the trade-offs when applying machine learning algorithms on different knowledge areas in SE. Instead, our taxonomy aims to support practitioners and researchers in understanding the high-level aspects and the impact of their AI applications. That decision is more on the strategic level rather than a solution instance (e.g., which ANN to choose from).

In general we agree with Harman that compartmentalising and deconstructing AI for SE into sub-domains is tempting but would be a mistake [24]. The SE community can benefit significantly by discussing the strategies rather than the solutions themselves, and we believe that AI-SEAL can help in this endeavour.

6 CONCLUDING REMARKS

In this paper we propose the AI-SEAL taxonomy to help researchers and practitioners to classify different AI applications in software engineering. The taxonomy has three facets allowing its users to classify, the point of application (process, product and runtime), the type of AI technology (based, initially, on the five tribes proposed by Domingos [15]) and the level of automation of the applied technology (inspired by Sheridan-Verplanck's 10 levels of automation [41]).

Besides helping its user to understand the field of AI applications in SE, AI-SEAL provides a basis for software engineers to consider the risks of applying AI. This advantage allows, for instance, practitioners to reason about the trade-offs of introducing the AI technology in their processes and products. In addition, the taxonomy is not constrained by domain-specific applications, thus covering all knowledge areas of software engineering.

We demonstrate the use of the taxonomy by classifying 15 papers from past RAISE workshops. Most of the papers focused on supporting stakeholders during the development process but did not directly affect the source code or the runtime behaviour of the systems. There was also an uneven use of the many different AI approaches that exist; in particular a lack of Bayesian and, surprisingly, Connectionist (neural net) ones. Future work includes classification of more papers as well as proposing more detailed and explicit support for risk analysis.

[5] We also see a potential to analyse the overall 'reach' of the decisions/proposals of AI component in the wider system in a more detailed risk analysis. Since the ultimate risk might not be high, despite a high LA value, if the impact of the decisions are limited by the rest of the system.

REFERENCES

[1] W. Afzal and R. Torkar. 2008. A comparative evaluation of using genetic programming for predicting fault count data. In *3rd International Conference on Software Engineering Advances*. IEEE, NJ, USA, 407–414. https://doi.org/10.1109/ICSEA.2008.9

[2] W. Afzal and R. Torkar. 2008. A comparative evaluation of using genetic programming for predicting fault count data. In *3rd International Conference on Software Engineering Advances*. IEEE Press, NJ, USA, 407–414. https://doi.org/10.1109/ICSEA.2008.9

[3] S. Akbarinasaji, A. B. Bener, and A. Erdem. 2016. Measuring the principal of defect debt. In *5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '16)*. ACM, NY, USA, 1–7. https://doi.org/10.1145/2896995.2896999

[4] A. Barr and E. A. Feigenbaum. 1981. *The handbook of artificial intelligence*. HeirisTech Press, CA, USA.

[5] D. Barstow. 1987. Artificial intelligence and software engineering. In *9th International Conference on Software Engineering (ICSE '87)*. IEEE Computer Society Press, CA, USA, 200–211.

[6] M. Bäumer, P. Seidler, R. Torkar, R. Feldt, P. Tomaszewski, and L.-O. Damm. 2008. Predicting fault inflow in highly iterative software development processes: An industrial evaluation. In *19th International Symposium on Software Reliability Engineering: Industry Track*. IEEE, NJ, USA.

[7] B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. A. Brubaker, J. Guo, P. Li, A. Riddell, et al. 2016. Stan: A probabilistic programming language. *Journal of Statistical Software* 20, 2 (2016), 1–37.

[8] D. Charte, F. Charte, S. García, M. J. del Jesus, and F. Herrera. 2018. A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines. *Information Fusion* 44 (2018), 78–96. https://doi.org/10.1016/j.inffus.2017.12.007

[9] D. A. Clifton, J. Gibbons, J. Davies, and L. Tarassenko. 2012. Machine learning and software engineering in health informatics. In *1st International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '12)*. IEEE Press, NJ, USA, 37–41. https://doi.org/10.1109/RAISE.2012.6227968

[10] J. Davis, J. Hoffert, and E. Vanlandingham. 2016. A taxonomy of artificial intelligence approaches for adaptive distributed real-time embedded systems. In *2016 IEEE International Conference on Electro Information Technology (EIT)*. IEEE, NJ, USA, 233–238. https://doi.org/10.1109/EIT.2016.7535246

[11] F. G. de Oliveira Neto, R. Feldt, R. Torkar, and P. Machado. 2013. Searching for models to evaluate software technology. In *1st International Workshop on Combining Modelling and Search-Based Software Engineering*. IEEE, NJ, USA, 12–15.

[12] T. de Souza Alcantara, J. Denzinger, J. Ferreira, and F. Maurer. 2012. Learning gestures for interacting with low-fidelity prototypes. In *1st International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '12)*. IEEE Press, NJ, USA, 32–36. https://doi.org/10.1109/RAISE.2012.6227967

[13] T. Diamantopoulos and A. Symeonidis. 2015. Towards interpretable defect-prone component analysis using genetic fuzzy systems. In *4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '15)*. IEEE Press, NJ, USA, 32–38. http://dl.acm.org/citation.cfm?id=2820668.2820677

[14] S. M. Didar Al Alam, M. R. Karim, D. Pfahl, and G. Ruhe. 2016. Comparative analysis of predictive techniques for release readiness classification. In *5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '16)*. ACM, NY, USA, 15–21. https://doi.org/10.1145/2896995.2896997

[15] P. Domingos. 2015. *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books, NY, USA. https://books.google.se/books?id=glUtrgEACAAJ

[16] R. Feldt. 1998. Generating multiple diverse software versions with genetic programming. In *24th Euromicro Conference*, Vol. 1. IEEE, NJ, USA, 387–394.

[17] R. Feldt. 1999. Genetic programming as an explorative tool in early software development phases. In *1st International Workshop on Soft Computing Applied to Software Engineering*. IEEE, NJ, USA, 11–19.

[18] R. Feldt. 2014. Do system test cases grow old?. In *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*. IEEE, NJ, USA, 343–352.

[19] R. Feldt and S. Poulding. 2013. Finding test data with specific properties via metaheuristic search. In *24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, NJ, USA, 350–359.

[20] J. Frohm. 2008. *Levels of automation in production systems*. Ph.D. Dissertation. Chalmers University of Technology, Gothenburg, Sweden.

[21] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. 2014. *Bayesian data analysis*. Vol. 2. CRC press, FL, USA.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Vol. 27. Curran Associates, Inc., NY, USA, 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[23] M. Hamza and R. J. Walker. 2015. Recommending features and feature relationships from requirements documents for software product lines. In *4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '15)*. IEEE Press, NJ, USA, 25–31. http://dl.acm.org/citation.cfm?id=2820668.2820675

[24] M. Harman. 2012. The role of artificial intelligence in software engineering. In *1st International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '12)*. IEEE Press, NJ, USA, 1–6. https://doi.org/10.1109/RAISE.2012.6227961

[25] C. Heitmeyer, M. Pickett, L. Breslow, D. Aha, J. G. Trafton, and E. Leonard. 2013. High assurance human-centric decision systems. In *2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '13)*. IEEE Press, NJ, USA, 35–41. https://doi.org/10.1109/RAISE.2013.6615202

[26] M. Iliev, B. Karasneh, M. R. V. Chaudron, and E. Essenius. 2012. Automated prediction of defect severity based on codifying design knowledge using ontologies. In *1st International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '12)*. IEEE Press, NJ, USA, 7–11. https://doi.org/10.1109/RAISE.2012.6227962

[27] I. Keivanloo and J. Rilling. 2012. Clone detection meets semantic web-based transitive closure computation. In *1st International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '12)*. IEEE Press, NJ, USA, 12–16. https://doi.org/10.1109/RAISE.2012.6227963

[28] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. 2017. The case for learned index structures. *ArXiv e-prints* (Dec. 2017). arXiv:cs.DB/1712.01208

[29] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350, 6266 (2015), 1332–1338. https://doi.org/10.1126/science.aab3050 arXiv:http://science.sciencemag.org/content/350/6266/1332.full.pdf

[30] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. 2017. Building machines that learn and think like people. *Behavioral and Brain Sciences* 40 (2017), e253.

[31] M. Landhaeusser and R. Hug. 2015. Text understanding for programming in natural language: Control structures. In *4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. IEEE, NJ, USA, 7–12. https://doi.org/10.1109/RAISE.2015.9

[32] F. Langer and E. Oswald. 2014. A self-learning approach for validation of communication in embedded systems. In *3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2014)*. ACM, NY, USA, 38–44. https://doi.org/10.1145/2593801.2593808

[33] G. Marcus. 2018. Deep learning: A critical appraisal. *ArXiv e-prints* (Jan. 2018). arXiv:cs.AI/1801.00631

[34] J. Misra, S. Sengupta, and S. Podder. 2016. Topic cohesion preserving requirements clustering. In *5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '16)*. ACM, NY, USA, 22–28. https://doi.org/10.1145/2896995.2896998

[35] M. Monperrus. 2018. Automatic software repair: A bibliography. *ACM Comput. Surv.* 51, 1, Article 17 (Jan. 2018), 24 pages. https://doi.org/10.1145/3105906

[36] V. Musco, A. Carette, M. Monperrus, and P. Preux. 2016. A learning algorithm for change impact prediction. In *5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '16)*. ACM, NY, USA, 8–14. https://doi.org/10.1145/2896995.2896996

[37] P. Papadopoulos and N. Walkinshaw. 2015. Black-box test generation from inferred models. In *4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '15)*. IEEE Press, NJ, USA, 19–24. http://dl.acm.org/citation.cfm?id=2820668.2820674

[38] S. Roychoudhury, V. Kulkarni, and N. Bellarykar. 2015. Mining enterprise models for knowledgeable decision making. In *4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '15)*. IEEE Press, NJ, USA, 1–6. http://dl.acm.org/citation.cfm?id=2820668.2820670

[39] M. Schindler, O. Fox, and A. Rausch. 2015. Clustering source code elements by semantic similarity using Wikipedia. In *4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '15)*. IEEE Press, NJ, USA, 13–18. http://dl.acm.org/citation.cfm?id=2820668.2820672

[40] J. Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[41] T. B. Sheridan. 1980. Computer control and human alienation. *Technology review* 83, 1 (1980), 60–73.

[42] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff. 2004. NASA's swarm missions: The challenge of building autonomous software. *IT professional* 6, 5 (2004), 47–52.

[43] M. Unterkalmsteiner, R. Feldt, and T. Gorschek. 2014. A taxonomy for requirements engineering and software test alignment. *Transactions on Software Engineering Methodology* 23, 2, Article 16 (April 2014), 38 pages. https://doi.org/10.1145/2523088

[44] M. Usman, R. Britto, J. Börstler, and E. Mendes. 2017. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. *Information and Software Technology* 85 (2017), 43–59. https://doi.org/10.1016/j.infsof.2017.01.006