Department: SE for AI
Editor: Tim Menzies, timm@ieee.org

The Secret to Better AI and Better Software (is Requirements Engineering)

Nelly Bencomo, Jin L.C. Guo, Rachel Harrison, and Hans-Martin Heyn. Tim Menzies
Durham University, UK; McGill University, Canada; Oxford Brookes University, UK; University of Gothenburg, Sweden; NC State, USA

***Abstract*—Much has been written about the algorithmic role that AI plays for automation in SE. But what about the role of AI, augmented by human knowledge? Can we make a profound advance by combining human and artificial intelligence? Researchers in requirements engineering think so, arguing that *requirement engineering is the secret weapon for better AI and better software*.**

Much has been written about the algorithmic role that AI plays for automation in SE. But what about the role of AI, augmented by human knowledge? Can we make a profound advance by combining human and artificial intelligence? Researchers in requirements engineering think so, arguing that *requirement engineering is the secret weapon for better AI and better software*[1].

To begin, we first need a definition. What is requirements engineering or RE? RE used to be viewed as an early lifecycle activity that proceeded analysis, design, coding and testing. For safety critical applications there is certainly a pressing need to create those requirements before the coding starts (we will return to this point, later in the paper). However, in this age of DevOps and Autonomous and Self-adaptive systems, requirements can happen at many other times in a software project[15], [14]. We say that:

> *Requirements engineering is any discussion about what to build and how to trade-off competing cost/benefits. It can happen before, during, or after runtime.*

As shown in Table 1 and Table 2, there are many ways AI can help RE, across a broad range of SE activities. But, what about the other way around? If we add more requirements into AI, and use RE methods to get *truly desired* requirements, can we make better software by combining human and artificial intelligence?

In our view, when integrating AI into software engineering is a *co-design problem* between humans, the AI model, the data required to train and validate the desired behaviour, and the hardware running the AI model, in addition to the classical software components. This means that when integrating AI, you need to know and understand the context of the system in which you want to apply your AI model to derive the necessary model requirements [17].

For example, in the arena of safety critical systems, model construction must be guided by safety requirements. one challenge for AI in RE are safety standards that base on the EN-IEC 61508 standard[2]. These safety standards assume that for software only systematic faults exists. Therefore, they emphasise correct processes and the creation of lifecycle artifacts to minimise systematic mistakes during both the

[1]This paper is based on the Panel "Artificial Intelligence and Requirement Engineering: Challenges and Opportunities", which took place at the Eighth International Workshop on Artificial Intelligence and Requirements Engineering (AIRE).

[2]Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems; for example ISO 26262 for the automotive sector or IEC 61511 for the process industry.

Formal methods	Expressing domain knowledge and software as axioms, then reasoning about those acions.
Theorem provers	A special kind of formal methods. Finds settings to variables that satisfy constraints.
Data miners	Divide data into regions, summarize each region.
Defect predictors	A kind of data mining. Finds likely locations of bugs
Optimizers	Divide data into regions, then recommend how to move for worse to better regions.
Genetic algorithms	A special kind of optimizer that mutates populations of candidates over many generations (the best candidates are mutated and mixed together to create the next generation).
Text mining	Find structures in large sets of natural language.
Sentiment analysis	A special kind of text mining. Guess the mood of the an author, based on their writing.
Topic modeling	A special kind of text mining. Finds conjunctions of terms (called "topics") that often occur together in different parts of some text.

Table 1. Some AI tools useful for RE.

AI for RE at design time: One AI technology that can help software design work is text mining and, more specifically, sentiment analysis and topic modeling. Sentiment analysis can explore, at scale, many comments about certain apps and their concrete features. These tools can help app developers to systematically analyze user opinions about individual features, filter irrelevant reviews, and perform competitor analysis [1], [2]. As to topic modeling, this is a method for finding conjunctions of terms (called "topics") that often occur together in different parts of some text. Parker et al. use topic modeling to bridge the vocabulary gap between users' comments from the app review and product descriptions [3]. Stanik et al. adopt the latest language model to cluster topics from Twitter content about the software to reveal trends of the user feedback and therefore prioritize future effort. Apart from topic modeling, another AI technology that can help early life cycle discussions are multi-objective optimizers and theorem provers. These tools serve as aids to users as they navigate the competing constraints seen in software product lines. Theorem provers find settings to variables that satisfy constraints while multi-objective optimizers divide what we know about a domain into regions, then recommend how to move from worse to better regions. This can be useful during design discussions while struggling to navigate competing constraints in order to generate a design. Multi-objective optimizers know how to trade between competing concerns and are very useful– for example, see the work of Bowers et al. using search-based heuristics to address that issue [4]. See also Vescan et al. who use genetic algorithms to learn a tree of features that describe a current system [5] and Abdelnabi et al. who use natural language process techniques to generate object-oriented class diagrams from a requirements specification [6]. This technology is particularly exciting for requirements engineering since decades of research has resulted in optimization and theorem proving algorithms that scale to very large models (see, for example, Sayyad et al.'s exploration of 100,000s of requirements constraints in the Linux kernel [7]).
AI for RE during testing: AI is also useful for testing requirements and checking if the stated requirements are actually implemented. Automated traceability analysis builds on text mining and information retrieval methods to establish links between heterogeneous text documents or between textual requirements and source code [8], [9].
AI for RE during deployment: When we deploy software into the cloud, there is a need for automated "requirements agents" that can negotiate the appropriate access to shared resources. Backes et al. describe one such system deployed into Amazon Web Services [10]. After extracting theorems from the AWS software configuration files, that system runs theorem provers billions of times per day in order to ensure all the AWS access requirements are preserved.
AI for RE for configuration: Another requirements problem seen when a system is deployed is configuration. In their article 'Hey, You've Given me too Many Knobs!" [11], Tu et al. warn about what happens when users are giving too many options. They report that the overwhelming majority of configuration options are not used in modern software. This means, in turn, that that software is not fulfilling its requirements, e.g. minimizing energy usage, maximizing transfer of data, responding to user queries in a timely manner, etc. Here too, AI can proof to be very useful. Kaltenecker et al. report on the success of data mining technology to sample the effects of past configurations in order to suggest better configurations for now and the future [12].
AI for RE at runtime: During runtime several AI techniques offer potential to use data that is available just at runtime and that can support decision making, in order to deal with uncertainty in a principled way. For example, Bayesian inference and Bayesian surprises can be used for quantitative analysis to measure degrees of uncertainty and deviations of self-adaptive systems from normal behavior [13]. The key idea is that a surprising event can be defined as one that causes a large divergence between the belief distributions prior to and posterior to the event. Therefore, in order to satisfy its requirements, the system may decide either to adapt accordingly or to flag that an abnormal situation is happening during runtime, and based on data that was not available during design time. One research community that actively explores "AI for RE at runtime" is self-adaptation and autonomous systems. The authors of [14] argued that self-adaptive systems should be aware of their own requirements (i.e requirements happen at runtime). Requirements for self-adaptive systems are defined before runtime taking into account uncertainty [15], [16]. However, these requirements should also be run-time entities that can be reasoned over to understand the extent to which they are being satisfied, to support decision making for self-adaptation under uncertainty at runtime.

Table 2. Some examples of AI for RE across the life-cycle. Just to say the obvious, the above is just a small sample of where AI is being applied to RE. For the reader interested in the keeping up with the state of the art in this exciting area, we refer them to the annual proceedings of the International on Requirements Engineering Conference https://requirements-engineering.org.

	Challenge	Opportunity
Formal methods	Where to get the axioms?	Infer from config files
Text mining.	Sub-optimal	Hyperparameter optimization
Configuration.	It works. Why not used more?	More AI Education
Multi-objective optimization	Taming runtimes.	Use stakeholder knowledge to reduce search space towards Human-AI Collaboration. Common vocabulary and common understanding between different communities
Deep Learning	Deep confusion	
Other AI algorithms	Needs scaling; needs explanation	

Table 3. Some challenges and opportunities for the technology discussed above. Note that for the last three challenges, stakeholder knowledge is the key opportunity.

concept and the design phase of the system. Applying AI for RE in the concept phase would require, a safety qualified AI performing the RE tasks. Such "tool qualifications", for example for ISO 26262, would require that the AI tool is developed according to a suitable standard, or that evidence is proved ”*that the assessed tool errors either do not occur or will be detected*”[18]. That is, for such systems, humans are still needed to ensure we produce ethical AI [19].

This is boon and bane at the same time. On one hand it might be difficult and not feasible to consider every contextual attributes and their possible changes at design time for the entire operational environment of your system, see for example [15]. On the other hand, a clear understanding of context and constraints offers opportunities to simplify the learning problem, such that the data required to train the desired behavior can be found, or created, more efficiently. For example, assume that you want a simple classifier for handwritten numbers from 0 to 9. Traditionally, you would have to collect thousand of examples for each number to train a (deep learning) classifier, which will use properties such as curvature, shape, etc. to discriminate between the digits. Now assume you have the context that each number is written in a different, digit-specific color. In this context, the classifier only needs to learn the correlation between color and digit, which will require significant less data samples (specifically 10).

Safety critical systems must be understandable and auditable by humans (this is especially true if those systems ever fail and we must fix their problem). The more an AI tool understands human requirements, and human reasoning, the more they can produce simpler and more explicable models.

One concern with this "keep it simple" approach is that some fear that such simpler models will perform worse than more complex options. In some domains, this is certainly true; e.g. when processing 10,000 wavelets from a signal processing model which, in turn, is controlling an autonomous car. But in many domains, there is much evidence that simpler need not be more stupid; e.g. see the examples listed by Rudin [20] or recent results in software defect prediction [21]). In addition, high safety integrity demands systems that are as simple as possible; trying to achieve a high safety integrity level on a complex system will most likely cost a fortune, or is simply impossible.

The Road Ahead

Table 3 lists some of the technologies discussed above and their challenges. For example, until just recently, formal methods needed humans to manually craft the axioms needed for their operation. But now, we can apply formal methods to requirements engineering by automatically extracting those axioms from configuration files [10].

As to text mining (and, indeed, any AI inference tool), these came with so many configuration options that industrial practitioners routinely shipped to industry suboptimal products [22]. But using multi-objective optimizers, we can auto-configure and improve these complex tools [22].

Note that we mentioned configuration in both the last two paragraphs. There is so much recent success in automatic configuration in order to satisfy numerous requirements. This is partly due to the advent of the alignment of ML and DevOps in MLOps, which helps to increase automation [23]. So why are not these tools used more widely? The answer, we think is to improve AI education. Too many software engineers seem to view AI as an incomprehensible black box. We need to show software engineers that AI software is just software. That is, up to a point, software is something that can be can be refactored and improved by software engineers.

The last three rows of Table 3 deserve special attention. All these tasks must explore complex spaces to return, perhaps, very complex models. Here, we assert there is much benefit in injecting requirements to reduce the complexity and increase the explainability of these tools. The standard workhorse of industrial AI is *classification* and *regression* that forms models from independent variables to predict for a single class variable. Viewed from the lens of requirements engineering, that may seem a very strange process (at least, to non-RE people). The lesson from decades of requirements research is that users come from different stakeholder groups and the requirements of each group are expressed as different sets of multiple goals. That is, the *landscape* of the problem is not one hill with a single top. Rather, our stakeholders live in different high mountain valleys, surrounded by a intricate multi-dimensional landscape. An AI tool that tries to build one model with one goal across that landscape will end up satisfying very few of the stakeholders (if indeed, any at all). So what we propose is to reverse the formulation of the problem and consider what happens if we add requirements into AI:

- When an AI tool is aware of different requirements, they dramatically reduce their search space.
- Specifically, if we sort candidate solutions into different goals, and if we discard the worst half, then N-goals wipes out 2^N parts of the problem (caveat: and different stakeholder goals wipe out different parts of the space).
- Such requirements-aware tools can terminate in logarithmic time (orders of magnitude faster than alternate tools that do not exploit the landscape of the solution) [21].

An Interdisciplinary Approach

Interdisciplinary collaboration is critical. Unfortunately our sub-communities are divided by the languages we use. Unless we agree on common terminology, misunderstandings will persist. We think RE education (and SE in general) needs to improve. AI experts need to understand the importance of SE. A common vocabulary and understanding are needed for these communities, to be able to exploit their strengths [24]. Therefore, additional effort on educating AI experts the SE knowledge is also critical to bridge the communication gap and to set reasonable expectations.

Final Thoughts

In our view, intelligence means navigating a landscape of (often competing) constraints. And user requirements is how we define that landscape. That is, mathematically speaking, user requirements are the forces that change the shape of the ideas explored by AI[3]. Hence we say AI and RE are inexorably linked. We cannot get the most out of AI/RE without working more on RE/AI:

- AI techniques can offer useful comments on many parts of that landscape.
- Requirements engineering can be used to quickly rule out the less-than-useful parts of that space.

To say that another way, ignoring either can be detrimental to the other. And to get the most out of both each means working harder on each.

[3]For more on the mathematics of user requirements, see the formal analysis of Jureta et al. [25] or the empirical analysis of Mathew et al. [26].

REFERENCES

1. E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, 2014, pp. 153–162.
2. F. Dalpiaz and M. Parente, "Re-swot: from user feedback to requirements via competitor analysis," in *International working conference on requirements engineering: foundation for software quality*. Springer, 2019, pp. 55–70.
3. D. H. Park, M. Liu, C. Zhai, and H. Wang, "Leveraging user reviews to improve accuracy for mobile app retrieval," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015, pp. 533–542.
4. K. M. Bowers, E. M. Fredericks, R. H. Hariri, and B. H. Cheng, "Providentia: Using search-based heuristics to optimize satisficement and competing concerns between functional and non-functional objectives in self-adaptive systems," *Journal of Systems and Software*, vol. 162, p. 110497, 2020.
5. A. Vescan, A. Pintea, L. Linsbauer, and A. Egyed, "Genetic programming for feature model synthesis: a replication study," *Empirical Software Engineering*, vol. 26, no. 4, pp. 1–29, 2021.
6. E. A. Abdelnabi, A. M. Maatuk, T. M. Abdelaziz, and S. M. Elakeili, "Generating uml class diagram using nlp techniques and heuristic rules," in *2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2020, pp. 277–282.
7. A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Scalable product line configuration: A straw to break the camel's back," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013, pp. 465–474.
8. J. Cleland-Huang, O. Gotel, A. Zisman *et al.*, *Software and systems traceability*. Springer, 2012, vol. 2, no. 3.
9. M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: Automatically augmenting incomplete trace links," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 834–845. [Online]. Available: https://doi.org/10.1145/3180155.3180207
10. J. Backes, P. Bolignano, B. Cook, A. Gacek, K. Luckow, N. Rungta, M. Schaef, C. Schlesinger, R. Tanash, C. Varming, and M. Whalen, "One-click formal methods," *IEEE Software*, vol. 36, no. 06, pp. 61–65, nov 2019.
11. T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadker, "Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 307–319. [Online]. Available: https://doi.org/10.1145/2786805.2786852
12. C. Kaltenecker, A. Grebhahn, N. Siegmund, and S. Apel, "The interplay of sampling and machine learning for software performance prediction," *IEEE Software*, vol. 37, no. 4, pp. 58–66, 2020.
13. N. Bencomo and A. Belaggoun, "A world full of surprises: Bayesian theory of surprise to quantify degrees of uncertainty," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 460–463. [Online]. Available: https://doi.org/10.1145/2591062.2591118
14. P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for re for self-adaptive systems," in *2010 18th IEEE International Requirements Engineering Conference*, 2010, pp. 95–103.
15. A. J. Ramirez, A. C. Jensen, and B. H. C. Cheng, "A taxonomy of uncertainty for dynamically adaptive

systems," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, jun 2012, pp. 99–108. [Online]. Available: http://ieeexplore.ieee.org/document/6224396/

16. J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel, "RELAX: a language to address uncertainty in self-adaptive systems requirement," *Requirements Engineering*, vol. 15, no. 2, pp. 177–196, Jun. 2010.
17. A. Knauss, D. Damian, X. Franch, A. Rook, H. A. Múller, and A. Thomo, "Acon: A learning-based approach to deal with uncertainty in contextual requirements at runtime," *Information and Software Technology*, vol. 70, pp. 85–99, 2016.
18. International Organization for Standardization, *ISO 26262:2018: Road vehicles — Functional safety*. Geneva: International Organization for Standardization, 2018. [Online]. Available: www.iso.org
19. I. Ozkaya, "Ethics is a software design concern," *IEEE Software*, vol. 36, no. 3, pp. 4–8, 2019.
20. C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.
21. D. Chen, W. Fu, R. Krishna, and T. Menzies, "Applications of psychological science for actionable analytics," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 456–467. [Online]. Available: https://doi.org/10.1145/3236024.3236050
22. R. Krishna, Z. Yu, A. Agrawal, M. Dominguez, and D. Wolf, "The "bigse" project: Lessons learned from validating industrial text mining," in *Proceedings of the 2nd International Workshop on BIG Data Software Engineering*, ser. BIGDSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 65–71. [Online]. Available: https://doi.org/10.1145/2896825.2896836
23. S. Alla and S. Adari, "What is mlops?" 2021.
24. D. Piorkowski, S. Park, A. Y. Wang, D. Wang, M. Muller, and F. Portnoy, "How ai developers overcome communication challenges in a multidisciplinary team: A case study," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, no. CSCW1, Apr. 2021. [Online]. Available: https://doi.org/10.1145/3449205
25. I. Jureta, J. Mylopoulos, and S. Faulkner, "Revisiting the core ontology and problem in requirements engineering," in *2008 16th IEEE International Requirements Engineering Conference*, 2008, pp. 71–80.
26. G. Mathew, T. Menzies, N. A. Ernst, and J. Klein, ""short"er reasoning about larger requirements models," 2017.