Designing Personalized Learning Environments through Monitoring and Guiding User Interactions with Code and Natural Language

Mircea Lungu
mlun@itu.dk
Department of Computer Science
IT University of Copenhagen
Denmark

ABSTRACT

Learning the vocabulary of a new language and a new programming API are similar in multiple ways. In this paper we evaluate several of the similarities and show that based on them we can design systems that can guide the learner towards improving their knowledge without an external tutor or preset curriculum. Instead, the class of systems we propose here are based on automated approaches of building maps of knowledge of the domain by mining repositories. By intersecting this knowledge with models of learner knowledge built by observing past learner interactions with artifacts of the domain we can generate highly personalized learning guidance.

CCS CONCEPTS

• **Applied computing → Interactive learning environments**;
• **Software and its engineering → Software architectures**.

KEYWORDS

education, software engineering, architectures, ecosystems, artificial intelligence

ACM Reference Format:
Mircea Lungu. 2019. Designing Personalized Learning Environments through Monitoring and Guiding User Interactions with Code and Natural Language. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

From Facebook, to Google, Instagram and Twitter, we see that technology that provides personalized content generates very strong engagement. These highly addictive platforms do not create the content; instead, they enable users to find personally relevant content.

The entire personalization machinery, which involves complex algorithms and very extensive tracking of user interactions with information, is (unfortunately) mostly aimed at building user models to most efficiently deliver advertisements and selling anonymyzed data to interested third parties.

In this paper, we argue for using the same techniques in the service of learning. Indeed, an approach based on finding highly relevant personal content that is interesting to the learner and at the same time in the zone of proximal development could and should be used to boost learning at least in specific domains of knowledge.

We focus on two domains of knowledge that offer similar affordances for personalized education: learning new programming APIs and learning the vocabulary of a language. We are nevertheless not the first to observe the similarity between the two domains; in recent work, Hindle et al [1] show that the same statistical models used in modeling language are good (actually even better) at modeling Java source code.

2 VOCABULARY LEARNING FOR CODE AND NATURAL LANGUAGE

In this work, we will focus on what we will call *declarative knowledge landscapes*, subsets of knowledge where one has to memorize tokens and their meaning. In programming, developers have to remember tokens that make up APIs; they include usually classes, methods, and their roles. In natural language, the learned tokens represent concepts in the world. In the remainder of this paper, we will sometimes refer to this kind of knowledge tokens as *knowledge atoms*.

In the remainder of this section we highlight several similarities between code and natural language knowledge that will represent the fundamental reasons for our proposal of using similar systems for enhancing learning in the two domains.

Atomic Nature of Learned Knowledge. In both cases, a significant component of the learning is the memorization of symbols: words for a foreign language learner and API elements for a software developer. Both these types of knowledge require the memorization of individual tokens of meaning. Although there is also a syntactic component in both the domains, we do not focus on it at the moment.

Interconnectedness of Individual Atoms. In linguistics, chunks are words that occur together frequently and that even have a special meaning together. In programming API protocols are groups of methods that are often used together as part of frequent API workflows. Both these types of co-occurrence can be automatically detected by data mining the corresponding corpora.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In linguistics some words are part of the same family. In programming various methods apply to the same data structure. In both the domains knowledge can be represented as graphs, with various relationships represented as edges between the various knowledge atoms.

Skewed Usage Distribution of the Atoms in Corpora. Another similarity is the highly skewed distribution of token usage in corpora. In both natural language texts and source code a few tokens appear much more frequently than others. This presents an opportunity for automatically estimating the relative importance of various tokens based on their frequency of occurrence in corpora.

Interactions With Individual Atoms Can be Tracked. In both the domains, individual interactions between users and knowledge atoms can be tracked since a vast amount of interactions happen digitally. Developer interactions with source code have been tracked in the past to help in the development process (e.g. in the work of Hattori et al.[2] or Soetens et al.[3]). In the context of language learning, we have implemented ourselves interceptors that allow the tracing of interactions with texts at the individual word granularity level [4].

Large Amounts of Knowledge Available Online. In both foreign language learning and programming education there is a virtually infinite amount of content online. Moroever this content is growing at a fast pace. News articles, blogs, movies and podcasts for language learning. Live programming sessions, open source systems, StackOverflow questions and answers, for programming knowledge.

3 A VISION

We envision a system can be designed that can observe the learners constantly in their interactions with knowledge landscapes, infer models of their evolving knowlege, and guide them in further encounters with materials in such a way as to optimize their learning.

The guided interactions can be part of explicit learning sessions (e.g. exercises generated based on past interactions with knowledge) or implicit learning sessions, in which the learner works on a real task which is at the appropriate difficulty level. By appropriate difficulty level the system should aim for the "zone of proximal development": challenging, but not too challenging. Indeed, within the sea of information available online, every learner should be able to find materials that are at the right difficulty level and interesting for them.

4 A CONCEPTUAL ARCHITECTURE

We propose an architecture based on our experience with prototyping a system for language learning that aims to implement the vision exposed in this paper. The implemented prototype tracks user interactions with words in foreign texts, generates personalized exercises based on past readings and recommends future readings that are filtered based on declared user interests [5].

We believe that the same principles that work in the domain of vocabulary acquisition, can be ported to the domain of programming knowledge (and potentially even beyond, but that is not the subject of this work).

Figure 1 shows an informal conceptual architecture of a system that implements our vision[1]. In the remainder of this section, we describe and discuss the various components in the figure.

Monitoring User Interactions with the Knowledge Domain. One of the cornerstones of the proposed architecture is the detailed monitoring user interaction with artifacts in the knowledge domain. Based on these interactions knowledge will be inferred, so the more interactions can be tracked, the more precise inferences will be possible.

In our work on vocabulary learning, we intercept the translation requests that readers issue while reading in their learned foreign language. To intercept translations we implemented both a browser extension that the learners can use to obtain translations while browsing the net and a dedicated web application that can be used also from mobile devices [5]. Besides individual interactions with words, we also track interactions with automatically generated exercises, and the duration of the reading sessions from which we can infer reading speed.

In the context of software knowledge, researchers have reported work on intercepting and tracking micro-interactions between the developer and the source code with the help of IDE plugins (e.g. [2], [6]). Such an infrastructure can represent the starting point for a system like the one described here.

Evolving Learner Knowledge and Interest Models. Based on the monitoring process, the learner knowledge and interest model receives a stream of events that represent user interactions with the knowledge landscape. These events can be used as input to machine learning models. Two examples:

(1) When a user clicks on a word, the probability of them not knowing the word is increased. In opposition, for the words around the translated one which were not translated, the probability of them being known is increased. If an exercise that tests a particular word is solved faster than in the past this means that the probability that the word is known increases [7].
(2) When a programmer hovers over a function to show it's documentation this event can result in increasing the probability that they know it since they just saw its definition. When the switch to the browser, before coming back to type an API call, the probability that they known the call can be decreased.

However, learner knowledge is not static, and by observing worker interactions with the knowledge landscape we can infer the changes in their knowledge based on their new behavior (a user who translated the word *Blauzungekrankenheit* in the past but stopped translating it, has probably learned it).

Moreover, learner knowledge models must also integrate knowledge about human forgetting [8] to create an even more precise model of evolving learner knowledge.

For the situations where possible, the interest of the learner should also be modeled. Based on their interactions with texts one can guess the interest in the topic. Thus, a language learner

[1]The architecture is a generalization of the one employed in the personalized reading platform in which the *domain knowledge model* and *learner knowledge model* are quite basic at the time of this paper (mid 2019) [5]

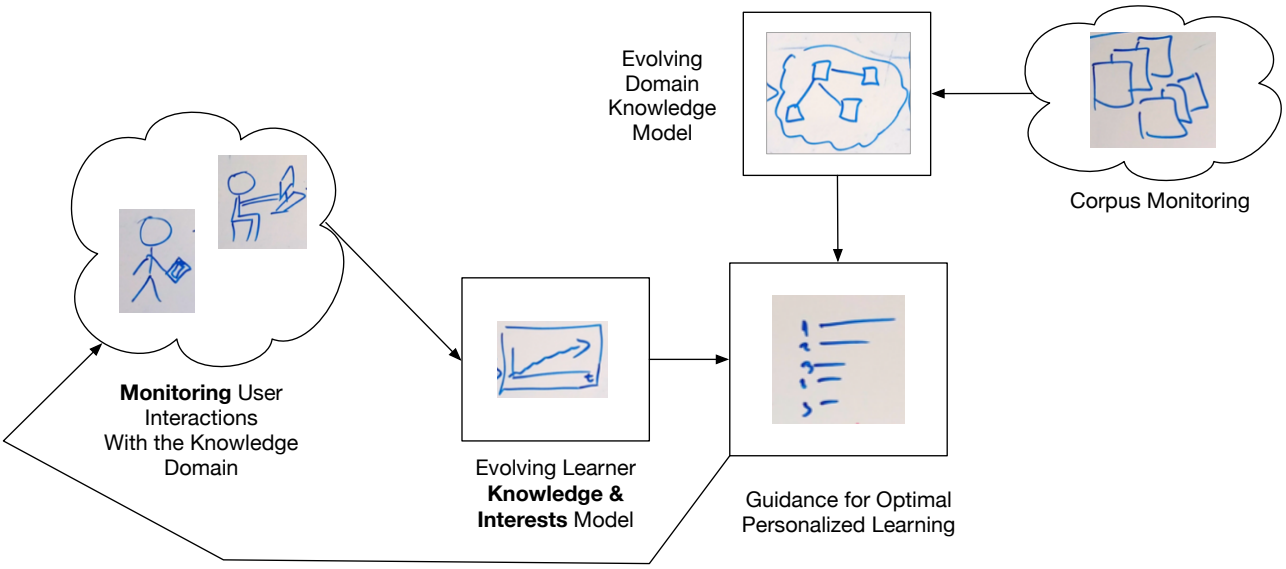

Figure 1: A Conceptual Architecture of the Proposed Personalized Learning Environment

interested in sport can be recommended more sport-related articles. And a coder interested in graphics can be recommended more tasks related to graphics, and less about databases.

Evolving Domain Knowledge Model. The simplest domain knowledge models can be linear and can be based on the frequency of occurrence of various knowledge atoms, given that they are long tail distributed in both of the domains.

More complex models would take into account the trend in usage and popularity for every element[2], and the co-occurrence relationships between individual atoms of knowledge.

Even more complex models would take into account the relationships between the atomic elements of knowledge from the knowledge landscape.

Corpus Monitoring. The system must "keep an eye" on the state of the relevant knowledge corpus and constantly update it when new information appears. The more complete the view of the corpus, the better the guidance can be. This can be a resource consuming activity and architectural approaches that support big data crawling should be considered.

Guidance for Optimized Personalized Learning. Based on the individual learner model, and the domain knowledge model, the system should be able to provide two types of guidance:

- Explicit learning sessions such as Exercises that are based on spaced repetition as argued for by Robbes et al. [9].
- Recommendations for implicit learning sessions. In situations where there are *more realistic tasks than the learner can address*, prioritizing to optimize for the right difficulty and learning can be done; this is the case for reading or watching video material in a foreign language. It remains to be seen whether this is also the case for software engineering knowledge. We expect it to be because the magnitude of content in software engineering is also quite high and dynamic (blog posts, issues in an open source project that a learner is particularly apt for doing, screencasts, StackOverflow questions, etc.).

5 DISCUSSION

Importance of Implicit Learning. As mentioned, we already have a system that is monitoring learners reading in foreign languages and provides a certain degree of guided learning. The system is available online [3] and is used by language teachers in one high school and two universities. With our current prototype that recommends personalized practice (both interesting texts to read and exercises), we have seen that the students are very motivated by the possibility of reading materials that they find interesting. Surprisingly, preliminary data shows that they are much more interested in implicit learning via realistic practice (i.e. reading texts that are interesting to them) than explicit practice (i.e. doing exercises based on the past words they did not understand).

Approach Requires Intermediate Level. When we talk about learners in our vision, we refer to intermediate learners. They must be past the beginner level to find realistic tasks for the implicit learning sessions and for the monitoring to observe them in realistic scenarios. While for the beginners there are a plethora of approaches and tools, for intermediates interested in deliberate practice [10] there are few.

Beyond Learning Tokens. We focused our discussion in this paper on simple types of declarative knowledge because we see it as a first step towards quantifying more complex types of knowledge. Indeed, other more advanced types of knowledge can be modeled and learning can be supported. In their work, Alexandru et al.

[2] Although the two domains have different speeds, both are continuously evolving

[3] https://www.zeeguu.org

showed how to automatically detect whether developers are using idiomatic Python coding conventions [11].

Knowledge Models Beyond Education. Being able to monitor the current knowledge of learners can be useful for other goals besides learning. One such context would be identifying experts automatically. This can be useful in large organizations that need to map the knowledge of their staff at the fine-grained level. Another application could be personalized complexity estimation of source code. Indeed, until now, most of the "code complexity" measures proposed are always indepedent of the subject, and this is unrealistic.

A Bootstrapping Problem. The presented approach works for domains of knowledge that have sufficient materials available online such that a map of the importance of individual atoms and their relationships can be constructed automatically.

Privacy. Knowledge models can be considered personal data and thus must be used judiciously.

6 CONCLUSIONS AND FUTURE WORK

In this paper we have presented the vision of systems that can generate personalized recommendations for learners of vocabulary in foreign languages and software systems. Based on a prototype system that is monitoring learners reading in foreign languages and guiding their learning with exercises and further reading suggestions we have presented a general architecture for such a system.

REFERENCES

[1] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. On the naturalness of software. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 837–847, Piscataway, NJ, USA, 2012. IEEE Press.

[2] Lile Hattori, Mircea Lungu, and Michele Lanza. Replaying past changes in multi-developer projects. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pages 13–22, October 2010.

[3] Quinten David Soetens, Romain Robbes, and Serge Demeyer. Changes as first-class citizens: A research perspective on modern software tooling. *ACM Comput. Surv.*, 50(2):18:1–18:38, April 2017.

[4] Mircea F. Lungu. Bootstrapping an ubiquitous monitoring ecosystem for accelerating vocabulary acquisition. In *Proccedings of the 10th European Conference on Software Architecture Workshops*, ECSAW '16, pages 28:1–28:4, New York, NY, USA, 2016. ACM.

[5] Mircea Filip Lungu, Luc van den Brand, Dan Chirtoaca, and Martin Avagyan. As we may study: Towards the web as a personalized language textbook. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, page 338, 2018.

[6] Romain Robbes and Michele Lanza. Spyware: a change-aware development toolset. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 847–850, New York, NY, USA, 2008. ACM.

[7] Everett Mettler and Philip J. Kellman. Adaptive response-time-based category sequencing in perceptual learning. *Vision Research*, 99:111 – 123, 2014. Perceptual Learning – Recent advances.

[8] Hermann Ebbinghaus. Memory: A contribution to experimental psychology. *Annals of neurosciences*, 20(4):155, 2013.

[9] Romain Robbes, Mircea Lungu, and Andrea Janes. Api fluency. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '19, pages 97–100, Piscataway, NJ, USA, 2019. IEEE Press.

[10] K. Anders Ericsson, Ralf Th. Krampe, and Clemens Tesch-romer. The role of deliberate practice in the acquisition of expert performance. *Psychological Review*, pages 363–406, 1993.

[11] Carol V. Alexandru, José J. Merchante, Sebastiano Panichella, Sebastian Proksch, Harald C. Gall, and Gregorio Robles. On the usage of pythonic idioms. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2018, pages 1–11, New York, NY, USA, 2018. ACM.