Explorotron: An IDE Extension for Guided and Independent Code Exploration and Learning (Discussion Paper)

Yoshi Malaise
ymalaise@vub.be
Web & Information Systems Engineering Lab
Vrije Universiteit Brussel
Brussels, Belgium

Beat Signer
bsigner@vub.be
Web & Information Systems Engineering Lab
Vrije Universiteit Brussel
Brussels, Belgium

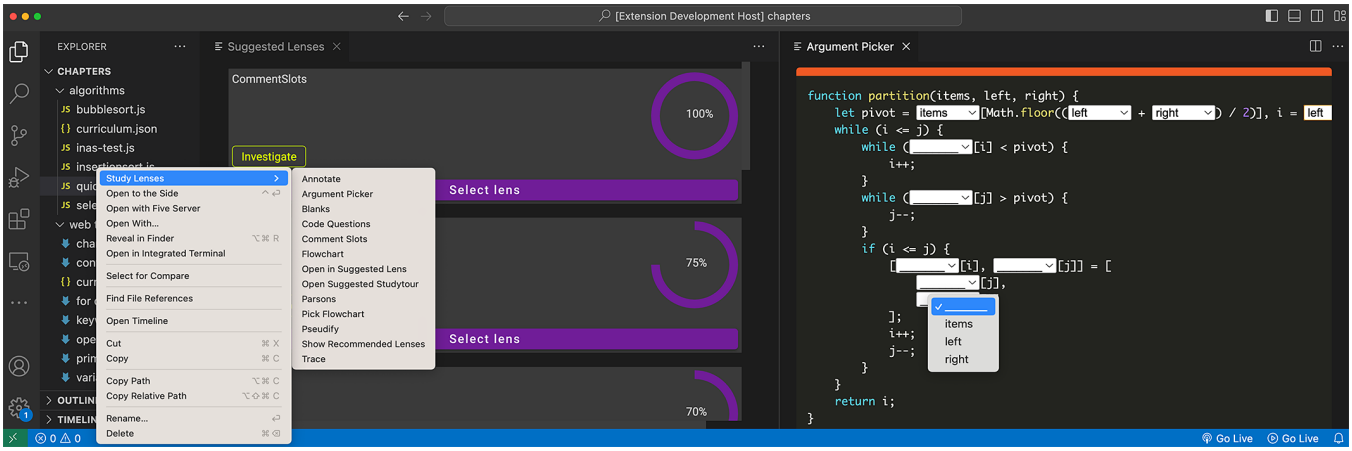


Figure 1: Explorotron Visual Studio Code extension showing recommended study lenses on the left and the *Argument Picker* study lens where students have to decide which argument goes where in the code on the right. Image altered due to space constraints.

ABSTRACT

We introduce the Explorotron Visual Studio Code extension for guided and independent code exploration and learning. Explorotron is a continuation of earlier work to explore how we can enable small organisations with limited resources to provide pedagogically sound learning experiences in programming. We situate Explorotron in the field of Computing Education Research (CER) and envision it to initiate a discussion around different topics, including how to balance the optimisation between the researcher-student-teacher trifecta that is inherent in CER, how to ethically and responsibly use large language models (LLMs) in the independent learning and exploration by students, and how to define better learning sessions over coding content that students obtained on their own. We further reflect on the question raised by Begel and Ko whether technology should *"structure learning for learners"* or whether learners should *"be taught how to structure their own independent learning"* outside of the classroom.

Koli Calling '23, November 13–18, 2023, Koli, Finland

ACM ISBN 979-8-4007-1653-9/23/11…$15.00
https://doi.org/10.1145/3631802.3631816

CCS CONCEPTS

• **Applied computing** → **Interactive learning environments**.

KEYWORDS

Study Lenses, PRIMM, Programming Education

ACM Reference Format:
Yoshi Malaise and Beat Signer. 2023. Explorotron: An IDE Extension for Guided and Independent Code Exploration and Learning (Discussion Paper). In *23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23), November 13–18, 2023, Koli, Finland*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3631802.3631816

1 INTRODUCTION

The work in this paper is a continuation of earlier work that proposes a curriculum packager combining a web environment, a mobile application and micromaterials to empower educators in helping their students to study from code available online, even if the educational institution has limited available resources [18]. This self-guided approach is essential and has become more important in recent years as the shortage of technically qualified STEM people implies that there is also a shortage of STEM teachers and educators [11].

We start by motivating the need for our proposed VS Code extension and positioning our solution within the field of Computing Education Research (CER). We then provide some details about

Explorotron[1] and illustrate how it addresses the different pedagogical requirements. After presenting our preliminary version of the system, we highlight its current strengths and limitations to initiate a discussion about a number of points that we feel are worthwhile to investigate in future research.

1.1 Motivation and Background

The main motivation behind this work was our experience as educational coordinator for HackYourFuture Belgium, a daughter company of Open Knowledge Belgium and a small non-profit organisation providing coding classes for asylum seekers, refugees and migrants. As part of this role, we were responsible for keeping the curriculum up to date with the regional demands of the industry, while still providing a solid foundation to students so they could grow on their own. While operating in that context is definitely inspirational, it also comes with many challenges. An example of such a challenge is the fact that the students are not regular students following traditional classes during the day and working on coding homework during the evenings, but our classes rather had to be taught in a hybrid setup where students are studying the content on their own during the week and we would meet up once a week (Sundays) to cover the most challenging aspects of the week in a flipped classroom setting. This means that most of the time students are learning on their own or in collaboration with their classmates, making improving our students' self-reliance one of the main goals. This makes Begel and Ko's [3] question whether technologies should *"structure learning for learners"* or whether learners should *"be taught how to structure their own independent learning"* outside of the classroom relevant for our work. Our work unavoidably tries to do both, as on the one hand we try to offer as streamlined solutions as possible, making students "job-ready" in the shortest feasible time, but on the other hand, we have to acknowledge our limitations and note that we are unable to teach them the large variety of tools and frameworks they might encounter during their job hunt. The main deciding factor of these students' success is not their proficiency in an individual tech stack, but rather to what extent they are able to pick up new skills as required by the ever-changing industry expectations. Over time, our approach to teaching steadily changed from trying to teach the latest in-demand framework to teaching students how to read and learn from code they encounter, regardless of where they encounter it. This is one of the pillars in the design of our proposed Visual Studio Code (VS Code) extension. We would also like to note that the organisation receives help from many great volunteer developers and alumni who offer to lead the modules that are being taught. While these volunteers prove to be an indispensable resource, this approach also comes with some downsides. While a system is in place where volunteers can first be an assistant coach helping a lead coach before going on to lead their own modules, and coaches do get debriefings after classes and feedback, there is no way for the organisation to provide real teacher training. This means that the team of teachers consists mostly of professional developers or recent graduates who might have little to no pedagogical or teaching experience.

[1] https://wise.vub.ac.be/project/explorotron

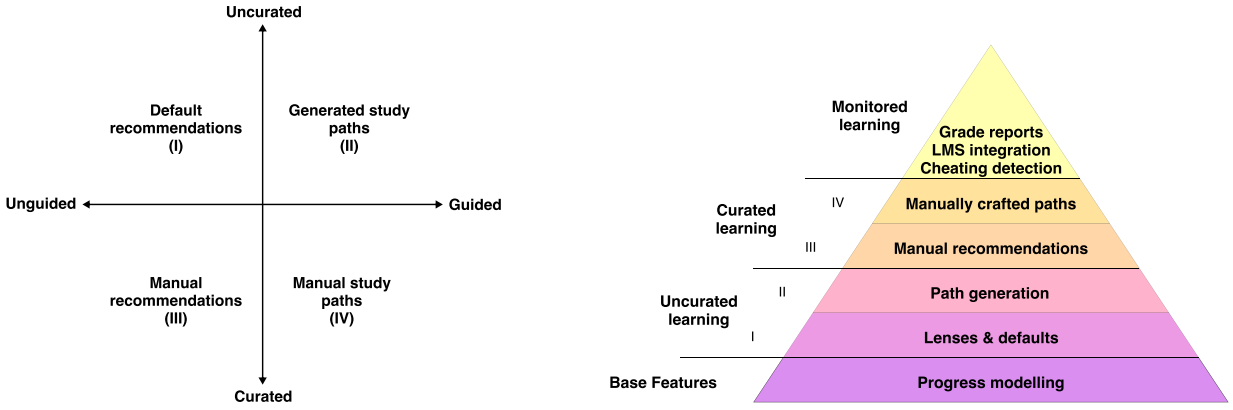
1.2 Situating in and Contributing to CER

In the following, we classify the different contributions of this paper according to the Translational Computing Education Research (TCER) model proposed by Cole et al. [9]. Many of the features presented in this paper fall in category 2A (theory-based designs) of the TCER model, where new interventions and tools are proposed for evaluation based on what we know from the literature. Furthermore, the whole setup is designed with computing education researchers in mind, so the setup of controlled empirical experiments (phase 2B) should be simplified by our approach. Further, we fully intend to open up the solution as an extendable baseline that steadily integrates solutions after they have proven to have a positive impact in phase 2 and researchers intending to work on more deepened practise research (phase 4) should see our solution as a viable option. The released artefact (the VS Code extension itself) also contributes to the learning process of individual learners who might not have access to traditional education. Finally, we hope that the discussion section of this paper might spark some interesting debates on decisions that need to be made when designing such a tool, which will hopefully turn into useful contributions to domain-specific theories in CER (phase 1B).

As illustrated in Figure 2a, we identify four contexts in which a learner can study code. On the y-axis, we make a distinction between content that has been specifically curated for the learner by educators and uncurated code. Curated content often offers tie-ins to other parts of the curriculum and is carefully crafted to provide logical steps with explanations and scaffolding. Uncurated content however is content the learner found somewhere (e.g. in a sample repository of a project they are interested in) but that might not have been specifically crafted with the goal of learning (or proper curriculum design) in mind. On the x-axis we make a distinction between whether the learner is freely exploring some code to satisfy their own curiosity or whether they are following a guided tour towards a specific goal. Plotting out the two axes results in four categories. Each of these categories benefits from different kinds of supportive technologies. However, we notice that these categories are not fully independent of each other. Even though things might be possible in a curated guided context (category IV) that would be completely off-limits in an uncurated and unguided context (category I), most things that benefit students in an uncurated and unguided context are also beneficial in the curated guided context. This led us to construct our tool in a pyramid fashion, where lower layers have to be built first to provide general support in all contexts, and as we rise higher towards more specific contexts, the tools become more specialised. Nevertheless, a learner can always make use of components defined in the underlying layers.

2 EXPLOROTRON PROTOTYPE

In the following, we describe the current state of our Explorotron VS Code extension for code study, as well as some parts that were already designed but are still under development. By discussing the current design, we aim to set the proper scene to have a fruitful discussion on future directions and open questions presented in the next section.



(a) Quadrants of learning

(b) Layering of learning tools

Figure 2: Overview of the different ways students can study code

2.1 Uncurated Unguided Learning (I)

As mentioned earlier, one of our main design goals is for students to be able to explore and learn from code regardless of where it comes from, which corresponds to quadrant I of our overview in Figure 2a. In order to accommodate this requested feature, we included a newly written implementation of the ideas described in study lenses [18], where there are functions that take a source code file as input and will generate a view on that file that focuses on learning or exploring certain aspects. We believe that a new implementation is necessary in order to better satisfy the *skill transfer* principle [8]. The principle of skill transfer states that learning one skill can have a negative or positive effect on how easy it will be to learn a new skill. But it also states that the context and environment in which a skill is learned become part of the skill acquisition. In order to gain optimal results, we should strive to teach students new skills in an environment as close to the realistic environment as possible without overwhelming them. This is why we opted for an extension that seamlessly integrates into the development environment that students are already using and to make as much use of pre-existing functionality and metaphors from that environment as possible. Users can simply start using study lenses to explore files by right-clicking on a file and choosing the lens they wish to apply from a context menu. Due to this seamless integration, we hope that the use of study lenses might become a natural part of the students' coding practice routine even after graduation and into their professional practice. Our current implementation contains the ten basic lenses described in the following, which can take any JavaScript file and help students discover certain aspects of the code.

Annotate Lens. Teaching has been proven to be an effective way for students to increase both their own understanding of the material and to increase their confidence [12, 32]. The annotate lens tries to support this process by providing the learner with a transparent whiteboard that is overlaid on the code (with highlighting applied). Students are free to highlight certain parts of the code and use markers to write, circle, underline or take notes. Students have mentioned that they like to use the tool when they are studying in groups with one person sharing their screen in a Microsoft Teams or Zoom call and explaining the code.

Argument Picker Lens. The argument picker is a lens based on the Predict, Run, Investigate, Modify and Make (PRIMM) approach to teach programming, stating that students should be asked to make modifications to existing code before being asked to write their own code from scratch [29]. This is said to reduce the stress expressed by students and leads to better overall results as shown in Section 2.2. In this lens, students are shown the functions that appear in the file they are studying, but all occurrences of any of the arguments in the function body have been replaced by drop-down lists from which a student has to select the right argument for every occurrence.

Blanks Lens. Blanks aims for the later stages of PRIMM. In this lens, students are shown the code they are studying, but random parts of the code have been replaced with underscores. It is then up to the students to restore the original snippet. Students can configure the difficulty level and which aspects to focus on to provide a more tailored exercise.

Code Questions Lens. The code Questions lens is heavily based on the work by Lehtinen et al. [15], where both static code analysis and an execution engine are used to generate questions about the code. The student will be presented with a set of generated multiple choice questions in a sidebar. After answering all the questions, the student will receive feedback on their answers.

Comment Slots Lens. Learning from worked examples has been proven to be a valid strategy [1]. In programming education, worked examples have been demonstrated to work if they are presented in conjunction with practice problems [24]. These worked examples make use of comments to indicate subgoals and label sections of the code. In the Comment Slots lens, we hook into this process by replacing all comments in the code field with drop-down lists. It is then up to the student to reconstruct the program by correctly figuring out which comment goes where.

Flowchart Lens. This lens will render the selected JavaScript file in the form of a flowchart. Learning how to read flowcharts has been argued in the past to be beneficial for students as it helps them

form a formal representation of coding concepts in their mind [28]. It is further useful to help students focus on the flow of a program instead of being distracted by language and syntax features.

Parsons Problem Lens. Parsons problems have been introduced as a way to assess a student's problem-solving skills through the code creation process in an easier-to-grade way than writing code from scratch and have since proven to be a good indication of a student's code writing capabilities [10]. In our Parsons Problem lens, students are presented with draggable code lines that will display a green check mark as soon as a line is placed in the correct position.

Flowchart Picker Lens. The Flowchart Picker lens builds on top of the Flowchart lens. Users will be presented the original code and three flowcharts. One of these flowcharts is the original flowchart matching the code and the other two flowcharts are minor variations generated by randomly adapting parts of the abstract syntax tree (by for example swapping branches or changing conditionals). It is up to the student to determine which of the three flowcharts matches the code.

Pseudocode Lens. Similar to flowcharts, pseudocode is used to help focus on the logic and flow of an implementation without having to worry about language details [21]. The pseudocode lens generates a pseudocode version of the code in the selected file. Studying the generated pseudocode can also support students in learning a shared notation that can improve collaboration.

Trace Lens. Research has shown that there is a strong correlation between code writing and code reading skills, with tracing being one way to improve the latter [17]. In the Trace Lens, students are presented with different types of trace tables on the right-hand side of their code editor. This way, they can practise tracing directly from within the integrated development environment (IDE). Students are also presented a `Trace Me` button that, when pressed, will evaluate the script and log every variable assignment/read that happens during the execution. Students can use the generated trace to check their manual trace tables when no teachers are available to correct their work.

Even though we are still targeting the uncurated and unguided situation, we can already provide some help to students when deciding which lens to use, as illustrated in the simple architecture shown in Figure 3. If the user would like to explore a JavaScript (JS) file but does not know which lenses to pick, they can select a '`Show Recommended Lenses`' option. As part of the applicability filter, the system will then evaluate a function for every lens to see whether the lens is applicable to the selected JS file. For example, if the user selected a JavaScript file with functions that take no arguments, it would not make sense to show the *Argument Picker* lens. The remaining lenses are then passed to the Ranking Engine which will assign each lens a score based on how relevant it is for the given file (e.g. the *Comment Slots* lens might work for a piece of code that only contains two comments, but it might not be the most interesting way to learn from the code). The student will then be presented with a sorted list of all applicable lenses with the most applicable lens at the top. They can also select '`Open in Suggested Lens`' in which case the system will skip the recommended lenses page and directly open the highest-ranked applicable lens.

2.2 Uncurated Guided Learning (II)

The lens recommendation system described in the previous section also forms the basis for the next quadrant (II) where we provide guidance in a code space that is not curated to be used for education. We would like to go beyond just recommending a single lens but rather suggest a full *study path* through a selected file by applying different lenses in order, following a logical progression.

There are many different ways to define logical progression. One could, for instance, base themselves on the Block model which provides a detailed overview of how a program can be understood, its blocks and the relations between them [27]. One could also follow some of the more general pedagogical taxonomies such as Bloom's taxonomy [6] or the SOLO taxonomy [5]. While all these approaches are interesting and definitely worthwhile to investigate in future work, for our current implementation we focus on the PRIMM methodology defined by Sentance et al. [29].

PRIMM (Predict, Run, Investigate, Modify and Make) is a way of structuring programming lessons in such a way that students gradually take ownership of code in a way that reduces their cognitive load. According to Sentance et al. [29], such lessons consist of the following parts: A *predict* stage in which the student focuses on how the code functions and predicts what it does, a *run* stage in which they will validate their prediction by running the code, an *investigate* stage in which the student answers questions about the code possibly aided with an activity such as annotating code, a *modify* stage in which the student starts to make modifications to existing code, varying from small changes to advanced new features and finally a *make* stage in which students should write a program from scratch using the same concepts and techniques that they have been practising in the earlier PRIMM stages.

The attentive reader might already have some ideas on how some of the lenses that we described earlier fit nicely into the different PRIMM stages. That is indeed the idea forming the basis of uncurated guided tours. Whenever a student indicates that they would like to start a study tour on a file that has not been curated, we will retrieve all relevant lenses as discussed in Section 2.1, select the most relevant ones for every PRIMM stage and generate a study session during which the student explores them in the appropriate order. An example of this in action could be a lesson plan that starts with the *Flowcharts* lens in the predict stage, then goes on to the *Trace* and the *Code Questions* lenses during the investigate stage, before presenting the *Parsons Problem* and *Blanks* lens in the modify stage. As more lenses are being added in the future, the study tour generation process can be expanded as well.

2.3 Curated Unguided Learning (III)

Up until this point, we have been discussing a situation in which the learner is studying code they found or wrote themselves but that has not specifically been curated for use with our tools. This is an important and integral part of our tool as it provides students the freedom to take their learning skills with them even if they move beyond the borders of our curriculum and it reduces the time needed to prepare the material. However, a lot more becomes possible when educators are also using our tools to generate content as illustrated in quadrant III. For instance, we have a *slideshow* view that uses

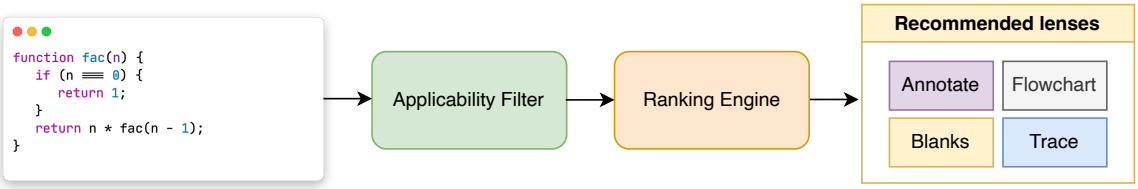


Figure 3: Overview of the architecture to generate the recommended lenses page

revealjs[2] to render markdown files as slideshow presentations. This can be a useful way for students to refresh some core concepts after a lecture. We also have dedicated views for teachers to create custom quizzes. These quizzes can contain multiple choice questions (both single and multiple answer variants) and code questions where students are presented an exercise (with optionally some starting code) and a set of unit tests that need to pass. Finally, we also allow the educator to indicate optimal lenses for specific files or directories. Whenever this happens and the students ask to open the suggested lens, it will pick the one the teacher configured and not the one the system would pick on its own.

2.4 Curated Guided Learning (IV)

When the teacher is not only designing individual pieces of content but in fact designing a whole course with clear steps that students will follow all the way through, that is when the content can be the most tailored towards a specific group (IV). This is inherently the case because more assumptions can be made about what students have already experienced when reaching a certain part. To support this use case, we designed and developed a study tour view that can be used by educators to construct a custom study tour over the material in the repository. They can do so by specifying which files should be studied in what order using which lenses, resulting in the generation of simple json configuration files with the custom `.study-tour` extension. These study-tour files can be checked into git and as such can become a part of the repository with educational content without needing an additional Learning Managing System (LMS). Whenever a student with an installed Explorotron VS Code extension tries to open the study-tour file, they will be shown an overview of all steps expected from them during that lesson and they will be presented these steps in sequential order, automatically moving on to the next one if the extension is able to see that an exercise has been completed successfully. Note that this mode of operation closely follows Biggs constructive alignment theory [4] as a teacher can select lenses that focus on the same concepts that they will focus on during future evaluations.

3 OPEN QUESTIONS AND FUTURE WORK

We have presented the current version of our Explorotron VS Code extension and its support for unguided or guided as well as uncurated or curated learning. However, there are still many open questions and possible future directions in which the presented research might evolve. In this section, we hope to initiate a debate resulting in some useful insights from the community on where to go from here.

[2]https://revealjs.com

3.1 How to find the right balance between stakeholders?

In our setup, we identify three main stakeholders. The *students* need to be able to use the tool to study the content they want to learn. *Teachers* need to be able to use the tool in such a way that they can—with little or no extra training—construct content and lesson plans to guide students to master a predefined set of learning goals, and *researchers* could gain a lot from using the extension as a data gathering tool and A/B testing environment to run trials or new Computing Education experiments.

Unfortunately, not all these goals are compatible. Let us for example take the case of the scientist and the student. As scientists, we would like to add a lot of data collection features to the Explorotron extension in order that we can learn from how students are using the tool. This data could include which lenses are used most often, whether students are using the generated study tour functionality, how often they are using the free exploration functionality as opposed to the guided learning functionality or the amount of time students are learning from external uncurated resources as opposed to the curated resources constructed by their teachers. However, from a student's perspective, these features are not desirable. Research has shown that electronic monitoring of job performance leads to decreased job satisfaction, increased stress levels as well as decreased autonomy and can be seen as a breach of trust [31]. One could argue that this is not a major problem if the data collection happens anonymously and is opt-in. However, this strongly depends on whether students trust the anonymity of the tool and there is a risk that the weaker students who stand to gain the most from the tool are the ones who might be least willing to share. At the other end of the spectrum, student self-assessment has been shown to lead to promising results [20], precisely because students feel in control, leading to improved motivation, engagement and learning. This self-assessment technique, however, does neither lead to insightful data that can be used by education researchers to improve future forms of learning, nor help teachers to know the level, the time/effort investment and the progress of their students that might be necessary if the taught course is supposed to lead to a degree or certification. This is why we would like to raise the question *"How ethical is it to potentially sacrifice some quality of the current generation of students by collecting as much data as possible, if by doing so we are more likely to have an increase in quality for the next generation of students?"*. The exact place on the axis between these two extremes is a point to be further discussed while continuing the work on our extension.

In a similar vein, we can look at the relationship between students and teachers. In our current design, we put the emphasis on the

learner rather than the teacher, by starting with the idea of the learner being able to use our tools even on content that is found from completely unrelated sources. We believe there is a lot of value in this approach, mainly because of the fact that students are not tied into a fixed ecosystem of learning but are able to take their ways of learning with them even after graduating. We hope that this can be a contributing factor to their lifelong learning and their self-reliance to keep up to date with the ever-changing technology market. Part of this open architecture stems from the fact that the extension can be installed for free by the student and no server infrastructure is required. But, this open-ended nature comes at a cost for teachers. Due to the lack of a centralised server, teachers might miss out on a lot of useful features of common learning management systems such as the ability to do timed submissions, the ability to see which students completed which portions of the module and general learner statistics. While we believe the trade-off is justified since a third-party LMS can still be used alongside our solution if necessary, that does not matter if teachers are unwilling to use our tool because they feel some of their needs are lacking. In the future, we might need to investigate whether it is worthwhile to explore a hybrid approach in which the extension can be used independently by students, but might also synchronise information with a server managed by an institution.

3.2 How to model users for a personalised learning experience?

Personalised learning has been heralded as one of the main contributions to improved learner agency, self-reliance and motivation [23]. This inspired a branch of research aiming to create intelligent tutoring systems making it possible to personalise education at scale. One way of using personalised learning to improve student outcomes is by adhering to the principle of expertise reversal. This principle states that when someone is learning a new skill, they should be provided with a lot of handholds and scaffolds of partial solutions so they can learn in a situation that maximises the intrinsic cognitive load and minimises any non-intrinsic cognitive load. It also states that a lot of the handholds that help beginning learners actually have a negative impact on more advanced learners [33]. Ideally, our lenses should be designed in a peel-away fashion such that instead of being one fixed lens, the lens itself can vary the amount of support it provides by looking at who is using it. Another way of implementing personalised learning is by looking at what content is suggested to be studied. In educational sciences, Vygotsky's zone of proximal development represents a zone of activities a learner would not be able to do on their own without any guidance (not too easy) but that are also not so difficult that they would not be able to do it even with proper guidance (not too difficult) [30]. According to this theory, one should always suggest exercises in this zone of proximal development as solving exercises that are too easy does not really lead to learning anything new and doing too difficult exercises will just lead to a feeling of failure and decreased satisfaction.

We would like to make more use of these principles, but in order to do so we will need to modify our extension to include some student modelling techniques. One possible representation of this knowledge is by providing a semantic representation of all topics and their relations within a given domain [25] in the form of a *knowledge graph*. For every topic in the graph, it is then possible to see how well the user performed and when some topics are too challenging and identify which underlying topics might be the cause of the problem. The use of these techniques both based on knowledge of domain experts and on machine learning have been studied in the past [13, 19, 22], but we need some *further investigation on how we can apply these concepts during study sessions on content that has not been curated*, presumably in the form of some automated assignment of the generated exercises to parts of the knowledge graph.

3.3 How to create suggested tours across a set of files?

While our current approach already helps to generate study tours to practise coding skills based on individual files, we lack an approach to help students study from a larger repository of content that has not been explicitly curated for learning. There is room for research towards techniques that can analyse a whole project and come up with a logical order to study specific elements of the project. This could take many shapes, part of it might be to detect which files follow similar patterns and suggest them together. Another approach might be to look at the dependencies within the files (which file imports which other files) and use that information to start a bottom-up tour. Alternatively, an approach could be used that employs machine learning to detect certain roles/functionalities in the code (e.g. controllers, views, business logic or logging) and tries to base tours on that structure. An additional challenge and/or opportunity could be to map the found code to parts of a knowledge graph for the domain of programming. This way, files could be recommended based on which topics they cover. Ideally, these path-generation techniques would hook into the student model as described above so that the student's performance in one tour can affect recommended future tours.

3.4 What is the role of Large Language Models in the presented approach?

In recent years, the field of artificial intelligence (AI) has made massive progress in the domain of Large Language Models (LLMs), as demonstrated by the appearance of tools such as ChatGPT[3], Bard[4] and GitHub Copilot[5]. These models took the world by storm and we are only now starting to see the first impact of these technologies on education. Becker et al. [2] discussed the potential impact, opportunities and challenges of LLMs in education. Research has been performed on how large language models might be used to generate more useful error messages that are easier understood by novice programmers so they can fix their code as they are learning [16]. Others have looked at ways to use large language models to automatically provide formative feedback to students when they submit code rather than solely relying on correctness checking via unit tests [14]. Chen et al. [7] took it one step further and embedded a programming tutor capable of providing code explanation directly in Visual Studio Code in the form of a ChatGPT-powered

[3] https://openai.com/blog/chatgpt
[4] https://bard.google.com/?hl=en
[5] https://github.com/features/copilot

extension. Sarsa et al. [26] followed a different approach by using the OpenAI Codex large language model to generate programming exercises, including solutions and test cases. From the generated samples, around 75% were classified as sensible, and 76.7% had a matching sample solution. However, only 30.9% of the solutions with unit tests actually passed all of these tests. Reviewing all the work that is done in this domain definitely gives us hope that they can be used as an amplifier in the future, and it would certainly be unwise to ignore them. However, we can also not overlook the fact that these approaches do make mistakes, sometimes come up with answers that can be nonsensical or confusing and sometimes even project a false sense of authority by pretending to cite sources that they made up. This can lead to negative impacts on the learning of students if they are attempting to study nonsensical content that might make fundamental mistakes. It raises the question *"to what extent it is ethical and pedagogically sound to include tools using these techniques in environments designed to teach students"*, a particularly important discussion to have in settings where students might go a long time on their own without receiving feedback from their teachers. What is the potential harm that can arise from the integration of such tools into our platform? One could imagine that tools such as clearer error messages might be considered less harmful when giving wrong results rather than code generation exercises. But even for the more automatic content generation approaches the answer is probably not binary. Some types of exercise generation should maybe not be exposed to students directly but could be offered to teachers as a way to scaffold exercises that still undergo revisions and a quality assurance process. Giving students access to these tools directly could be beneficial for their learning given that the context and expected errors are clear, but to answer these questions further research is needed. Further, we might also need to consider what the new role of the teacher will become; more emphasis may be put on the tasks of assessment and providing guiding feedback to discover new content.

4 CONCLUSION

We have presented our Explorotron VS Code extension prototype that helps students both in their guided and unguided exploration of code. We further described how Explorotron offers tools to deal with both content that has carefully been curated by educators as well as third-party content that the learners discovered themselves. To achieve this goal, our solution makes use of different code study lenses that present certain aspects of code files based on educational best practices, including PRIMM, the skill transfer principle and the principle of expertise reversal. We further discussed that for the next envisioned version of Explorotron, we need to answer a number of open questions such as *"how to find the right balance between stakeholders?"*, *"how to model users for a personalised learning experience?"*, "*how to create suggested tours across a set of files?*" and *"what is the role of Large Language Models in the presented approach?"*. The answers to these questions will help us in guiding the further design of Explorotron, making it an open and accessible learning support tool available to anyone who wants to learn to program through guided or independent code exploration.

ACKNOWLEDGMENTS

We would like to thank Evan Cole for his assistance and his explanations regarding the initial study lenses environment as well as his support in extending the curriculum he designed for Open Knowledge Belgium.

REFERENCES

[1] Robert K. Atkinson, Sharon J. Derry, Alexander Renkl, and Donald W. Wortham. 2000. Learning From Examples: Instructional Principles From the Worked Examples Research. *Review of Educational Research* 70 (2000). https://doi.org/10.3102/00346543070002181

[2] Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard-Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proc. of SIGCSE 2023*. https://doi.org/10.1145/3545945.3569759

[3] Andrew Begel and Amy J. Ko. 2019. *Learning Outside the Classroom*. Cambridge University Press. https://doi.org/10.1017/9781108654555.027

[4] John Biggs. 1996. Enhancing Teaching Through Constructive Alignment. *Higher education* (1996).

[5] John B Biggs and Kevin F Collis. 2014. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Academic Press. https://doi.org/10.1016/C2013-0-10375-3

[6] Benjamin S. Bloom. 1956. Taxonomy of Educational Objectives. Vol. 1: Cognitive Domain. *New York: McKay* 20, 24 (1956).

[7] Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. 2023. GPTutor: A ChatGPT-powered Programming Tool for Code Explanation. *arXiv preprint arXiv:2305.01863* (2023). https://doi.org/10.48550/arXiv.2305.01863

[8] Dan S Chiaburu and Sophia V Marinova. 2005. What Predicts Skill Transfer? An Exploratory Study of Goal Orientation, Training Self-Efficacy and Organizational Supports. *International Journal of Training and Development* 9, 2 (2005). https://doi.org/10.1111/j.1468-2419.2005.00225.x

[9] Evan Cole, Yoshi Malaise, and Beat Signer. 2023. Computing Education Research as a Translational Transdiscipline. In *Proc. of SIGCSE 2023*. https://doi.org/10.1145/3545945.3569771

[10] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a New Exam Question: Parsons Problems. In *Proc. of ICER 2008*. https://doi.org/10.1145/1404520.1404532

[11] Britton H Devier. 2019. Teacher Shortage and Alternative Licensure Solutions for Technical Educators. *The Journal of Technology Studies* 45, 2 (2019).

[12] Stanley Frager and Carolyn Stern. 1970. Learning by Teaching. *The Reading Teacher* 23, 5 (1970).

[13] Eleni Ilkou and Beat Signer. 2020. A Technology-enhanced Smart Learning Environment Based on the Combination of Knowledge Graphs and Learning Paths. In *Proc. of CSEDU 2020*. https://doi.org/10.5220/0009575104610468

[14] Charles Koutcheme. 2022. Towards Open Natural Language Feedback Generation for Novice Programmers using Large Language Models. In *Proc. of Koli Calling 2022*. https://doi.org/10.1145/3564721.3565955

[15] Teemu Lehtinen, André L Santos, and Juha Sorva. 2021. Let's Ask Students About Their Programs, Automatically. In *Proc. of ICPC 2021*. https://doi.org/10.1109/ICPC52881.2021.00054

[16] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *Proc. of SIGCSE 2023*. https://doi.org/10.1145/3545945.3569770

[17] Raymond Lister et al. 2004. A Multi-national Study of Reading and Tracing Skills in Novice Programmers. *ACM SIGCSE Bulletin* 36, 4 (2004). https://doi.org/10.1145/1041624.1041673

[18] Yoshi Malaise, Evan Cole, and Beat Signer. 2023. Codeschool in a Box: A Low-barrier Approach to Packaging Programming Curricula. In *Proc. of CSEDU 2023*. https://doi.org/10.5220/0011967900003470

[19] Yoshi Malaise and Beat Signer. 2022. Personalised Learning Environments Based on Knowledge Graphs and the Zone of Proximal Development. In *Proc. of CSEDU 2022*. https://doi.org/10.5220/0010998600003182

[20] James H McMillan and Jessica Hearn. 2008. Student Self-Assessment: The Key to Stronger Student Motivation and Higher Achievement. *Educational Horizons* 87, 1 (2008).

[21] Anne L Olsen. 2005. Using Pseudocode to Teach Problem Solving. *Journal of Computing Sciences in Colleges* 21, 2 (2005). https://doi.org/10.5555/1089053.1089088

[22] Gio Picones, Benjamin PaaBen, Irena Koprinska, and Kalina Yacef. 2022. Combining Domain Modelling and Student Modelling Techniques in a Single Automated Pipeline. *International Educational Data Mining Society* (2022). https://doi.org/10.5281/zenodo.6853131

[23] Vaughan Prain, Peter Cox, Craig Deed, Jeffrey Dorman, Debra Edwards, Cathleen Farrelly, Mary Keeffe, Valerie Lovejoy, Lucy Mow, Peter Sellings, et al. 2013. Personalised Learning: Lessons to be Learnt. *British Educational Research Journal*

39, 4 (2013). https://doi.org/10.1080/01411926.2012.669747

[24] Siti Soraya Abdul Rahman and Benedict du Boulay. 2010. Learning Programming via Worked-examples. *PPIG-WIP, Dundee 2010* (2010). https://doi.org/10.1016/j.chb.2013.09.007

[25] Mariia Rizun. 2019. Knowledge Graph Application in Education: A Literature Review. *Acta Universitatis Lodziensis: Folia Oeconomica* 3, 342 (2019). https://doi.org/10.18778/0208-6018.342.01

[26] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proc. of ICER 2022*. https://doi.org/10.1145/3501385.3543957

[27] Carsten Schulte. 2008. Block Model: An Educational Model of Program Comprehension as a Tool for a Scholarly Approach to Teaching. In *Proc. of ICER 2008*. https://doi.org/10.1145/1404520.1404535

[28] Andrew Scott, Mike Watkins, and Duncan McPhee. 2007. A Step Back From Coding: An Online Environment and Pedagogy for Novice Programmers. In *Proc. of JICC 2007*, Vol. 2007.

[29] Sue Sentance, Jane Waite, and Maria Kallia. 2019. Teaching Computer Programming With PRIMM: A Sociocultural Perspective. *Computer Science Education* 29, 2–3 (2019). https://doi.org/10.1080/08993408.2019.1608781

[30] Karim Shabani, Mohamad Khatib, and Saman Ebadi. 2010. Vygotsky's Zone of Proximal Development: Instructional Implications and Teachers' Professional Development. *English Language Teaching* 3, 4 (November 2010). https://doi.org/10.5539/elt.v3n4p237

[31] Rudolf Siegel, Cornelius J König, and Veronika Lazar. 2022. The Impact of Electronic Monitoring on Employees' Job Satisfaction, Stress, Performance, and Counterproductive Work Behavior: A Meta-Analysis. *Computers in Human Behavior Reports* 8 (2022). https://doi.org/10.1016/j.chbr.2022.100227

[32] John Spencer. 2003. Learning and Teaching in the Clinical Environment. *British Medical Journal* 326, 7389 (2003). https://doi.org/10.1136/bmj.326.7389.591

[33] John Sweller, Paul L Ayres, Slava Kalyuga, and Paul Chandler. 2003. The Expertise Reversal Effect. (2003). https://doi.org/10.1207/S15326985EP3801_4