# Investigating How Novices Use and Collaborate with a Version Control System for Block-Based Languages

Stefan Seegerer
*Computing Education Research Group*
*Friedrich-Alexander-Universität*
Erlangen-Nürnberg, Germany
stefan.seegerer@fau.de

Tilman Michaeli
*Computing Education Research Group*
*Friedrich-Alexander-Universität*
Erlangen-Nürnberg, Germany
tilman.michaeli@fau.de

Ralf Romeike
*Computing Education Research Group*
*Freie Universität*
Berlin, Germany
ralf.romeike@fu-berlin.de

***Abstract*—Collaboration is an important approach to computational thinking and considered a desired learning outcome in various computer science curricula. Collaboration in software projects is supported by the use of version control systems, in professional contexts, as well as in education. For novices, using a professional version control system poses major hurdles. To support the designing and teaching for and with version control systems, we conducted a case study with a didactically-adapted version control system. In a one-day workshop setting, we investigated how novices use a didactically-adapted version control system for block-based languages and how they collaborate when working with it. To this end, we applied a structural qualitative content analysis to screen recordings collected from the participants. Our results show that novices adapt most of the concepts of a version control system intuitively and that the usage of such a system can support their workflow. Nevertheless, some patterns (such as meaningful commit messages or using the tool to support tinkering) need to be addressed in teaching. Regarding collaboration, the results indicate that version control systems offer different forms of collaboration that might seem unnatural to students.**

***Index Terms*—Version Control Systems, Collaboration, Computer Science Education, Block-based Programming, High School, Novices**

## I. Introduction

Collaboration is considered as one of the most important soft skills in the 21st century (e.g. [1], [2]) and poses a core approach to computational thinking [3]. Working in groups, especially early on, provides a lot of advantages [4] and learners can profit from sharing or discussing their actions, reflecting and building on the work of others [5]. On the one hand, collaboration can have a beneficial effect on the process of learning (e.g. [6]), on the other hand, collaboration skills are also a desired learning outcome in various curricula (e.g. ACM/CSTA standards for K12 education [7] or the British Computing curriculum [8]). When working on software projects, students need to work jointly and collaboratively on the same resources. This poses challenges such as keeping documents up to date, distributing or merging changes, or handling conflicts resulting from changes to the same resources made by more than one person.

In professional software development, version control systems (VCS) are used to overcome this problem as they enable teams to work together on the same project by sharing common resources. In doing so, VCS allow for unique ways of collaboration. For example, conflict resolution makes it easier to work on the same resources in parallel. Therefore, work can be distributed in different ways. Furthermore, working in branches allows for tinkering in a risk-free environment, where progress from other members can easily be merged into the current branch.

However, these collaborative tools are not typically used in classrooms. On one hand, this is due to their complexity [9] and the fact that they are not well-suited for popular learning tools such as block-based languages [10]. On the other hand, students need to grasp the concepts related to a VCS to successfully use them for collaboration. To tackle the issue of complexity, in a previous paper, we presented a didactically-adapted version control system for the block-based language Snap*!* [11]. It was developed based on a review of existing professional and didactically-adapted version control systems and their use in computer science (CS) education. However, it remains unclear which concepts can be understood intuitively, and which ones must be explicitly addressed in teaching. Therefore, our aim is to understand how novices use the concepts of a VCS to support the design of didactically-adapted VCS and their usage in teaching. For the same reasons, we are interested in how students collaborate when supported by a version control system. Therefore, in this paper, we report on findings of a case study with year ten students (aged 15 to 16), who used a didactically-adapted version control system in an agile project setting.

The paper is structured as follows: Section II outlines the different kinds of collaboration on documents in computer science education and reports on experience with version control systems in the classroom. The didactically-adapted version control system that is used in the study is presented in section III. In section IV, we describe the qualitative methodology of our study. This section is followed by the presentation (section V) and discussion (section VI) of the results. In section VII,

we present our conclusions.

## II. Related Research

### A. Collaboration in CS Education

Roschelle and Teasly define "collaboration as the mutual engagement of participants in a coordinated effort to solve the problem together" [12]. Collaboration is an important aspect of working as a computer scientist, especially when working in software projects. It includes factors such as decomposition of tasks or communication among each other and promotes motivation and commitment (e.g. [13]).

Collaboration can be distinguished into *synchronous* and *asynchronous* activities [12]. Both variants can be found in Computer Science when *working together* on the same software project: on the same files at the same time (synchronous), and by deliberately splitting work into different tasks, which are then distributed among the team and need to be put together afterward (asynchronous).

Typically, collaboration on shared resources in CS is supported by certain tools. For *synchronous* collaboration, there are a lot of tools in education [14], [15]– even for block-based languages – like Netsblox [16] or Kanto [17], that allow for real-time remote collaboration, thus enabling distant pair programming. Synchronous collaboration, especially in the form of pair programming, has been subject of many studies (e.g. [18]–[24]). To foster collaboration in software projects in an *asynchronous* way, version control is widely adopted in professional contexts [25] and can also be used in classrooms [26].

### B. Version control systems in the classroom

Version control systems have been adapted for classroom settings both at school and university level [9], [26]–[30]. Using a VCS provides several use-cases and advantages in educational settings. These can be divided into pedagogical and organizational advantages.

Pedagogical advantages include easier collaboration, allowing for regularly merging partial programs – and therefore helping to identify interface problems at an early stage of the project –, the possibility to assess individual contribution, a visual development progress visible for the teacher and students, and data security in the sense of backups [28]–[31].

Version control systems also provide organizational advantages, such as the ease of posting assignments, giving feedback, providing skeletons, reverting changes, and working remotely. Furthermore, being able to access the code from everywhere, as well as distribute course materials or handle homework submissions are frequently mentioned [29], [30].

However, literature reports a series of problems connected with the use of version control systems. For example, novices tend to use a non-iterative workflow with long periods without a commit (i.e. adding changes to the VCS). This poses an obstacle especially for beginners and diminishes a lot of the benefits of using a VCS [28]. From a student's point of view, conflicts and their resolution are the most complex and difficult tasks [32], especially when the students always work in the centralized repository rather than in their own branches [29].

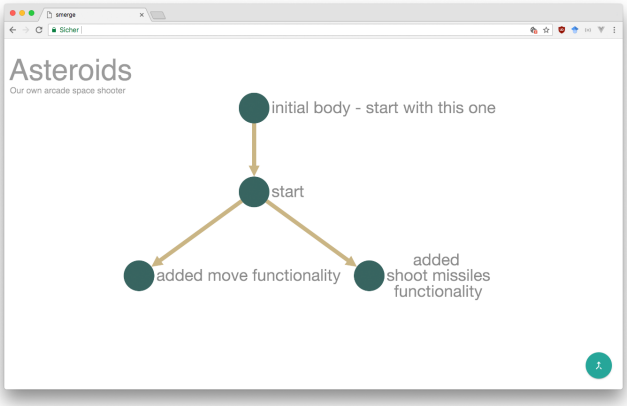


Fig. 1. Browser-based UI for a version control system for block based languages

Overall, professional version control systems are reported as being hard to learn (cf. [9], [32]) and the introduction of a professional version control system is associated with a large overhead. Students have to develop an understanding of the concepts of VCS and familiarize themselves with the respective commands as well as typical procedures. Even for entry-level professionals or graduate students, this is a complex (and difficult) task [9].

In summary, we see that both *synchronous* and *asynchronous* collaboration activities play a huge role in computer science. While the first one, especially in the context of pair programming, has been a common subject of research, there are only few studies regarding asynchronous collaboration and the interaction with a version control system in educational settings.

## III. A Version Control System for Block-Based Languages

VCS have been a feature block-based languages were missing [10]. The didactically-adapted version control system for block-based languages we presented previously is designed based on a review of existing professional and didactically-adapted version control systems and their use in CSE [11].

Our goal was to develop a VCS that is accessible to novices, but still includes all core concepts of a VCS, which are:

- *Project history*: Through versioning and awareness information such as time stamps or commit messages, a traceable project history becomes accessible.
- *Committing*: Changes, such as adding new files or altering existing ones, are added to the project by committing them to the version control system.
- *Branching*: Branching opens an alternative path so that changes can be made in parallel. A branch can be used for developing new features or fixing bugs, without impacting the current state of the project.

- *Merging*: VCS simplify combining changes and resolving conflicts automatically or – if necessary – manually.
- *Data backup*: Backing up all files under version control allows for returning to every (older) version, and, in consequence, risk-free tinkering.

Our version control system is realized as a web-tool. The project and its history are visualized in a graph. Changes are committed back to the VCS from within Snap*!*. For each editor of a version, a branch is created automatically. Changes are merged by selecting the respective nodes in the graph (see figure 2). Conflicts are resolved automatically when possible or visualized in a merge view. All old versions can be accessed via the graph visualization and are opened directly in Snap*!*.

## IV. Methodology

### A. Research Questions

This paper aims to investigate how the didactically-adapted version control system is used and how it affects collaboration among novices as this is important to develop suitable tools and teachings strategies for novices. Therefore, the following research questions are addressed in our case study:

- **(RQ1)** How do novices use a didactically-adapted version control system in a software project? Version control systems provide concepts such as committing, having a project history, branching, merging or data back-up. However, it is reported in literature that the use of a VCS poses major hurdles for novices. An understanding of the use of a didactically-adapted version control system can help tailor tools more to the needs of novices and indicates what teachers need to address in teaching.
- **(RQ2)** How do novices collaborate when they work with a didactically-adapted version control system? VCS support unique ways of collaboration. However, these ways might not be obvious to students as they differ from traditional collaboration in groups, and are therefore not used. Understanding how novices collaborate intuitively with a VCS supports teachers in addressing useful collaboration patterns, that are not intuitive for novices.

### B. Study Setting

The study was conducted during a workshop, where students implemented a project with Snap*!*. We applied an agile approach according to Romeike and Göttel [33], introducing practices such as project boards (see figure 3), user stories, and tasks.

The workshop was conducted with 18 students in a year ten (ages 15 to 16) class. They had computer science lessons and were familiar with terms from agile development such as user-stories or tasks, but had no experience with version control systems yet. Thus, they can be considered novices in working with VCS. The students were split into groups of 4 to 5 students with two programming pairs each, leading to a sample size of 8. Morse [34] recommends a sample size of about six for similar experiments.

To explore the natural way novices use a VCS, in the beginning, the students were given only a swift 5-minute introduction into the agile process in combination with Snap! and our version control system. As our goal was to identify which concepts can be understood intuitively, and which ones must be explicitly addressed in teaching, a short introduction is ideal to create a foundation for students to use our version control system in their projects. The workshop instructors started off by showing the project board (similar to the one in fig. 3). An exemplary task was then selected. The students followed along while the workshop instructors branched and merged two commits. Afterward, they were shown how to create their own projects in the version control system. Finally, the students chose their projects from the field of classic video games.

TABLE I
Tags applied to individual events in video recordings

| Category | Tag | Description |
|---|---|---|
| Interaction with Snap*!* | Tinkering | Trying new things without an explicit goal |
| | Coding | Creating a new sprite or script and trying different strategies to achieve a goal |
| | Transforming | Making changes to an existing sprite or script |
| | Designing | Creating or changing a sprite's costumes |
| | Testing | Assuring that code functions correctly |
| | Cleaning | Cleaning up the code, rearranging scripts |
| | Opening | Opening a project version in Snap*!* |
| | Closing | Closing a Snap*!* project |
| Usage of VCS | Exploring Project History | Accessing project information, checking for new nodes in the version control system, or opening versions for inspection |
| | Committing | Committing a new version to version control |
| | Branching | Creating a branch to work on a feature or solve a bug |
| | Merging | Combining Changes from multiple nodes and resolve conflicts |
| | Data Backup | Using an old version to compare for errors or returning to an old version |

### C. Data Analysis

To answer the research questions we mainly rely on video recordings, that we collected from each screen. We collected 4 $\frac{3}{4}$ hours of video per programming pair and 38 hours in total. In addition, we took the project board status and the final project structures from the version control system into account.

For the analysis of the data collected with screen recordings, we applied a structured content analysis approach according to Mayring [35]. The actual coding was conducted by two researchers using the software MaxQDA. To examine the way the students worked, we categorized all of their actions into a deductive category system (see table I) to identify recurring patterns. The category system comprised all possible interactions with Snap*!* as well as the core concepts of a

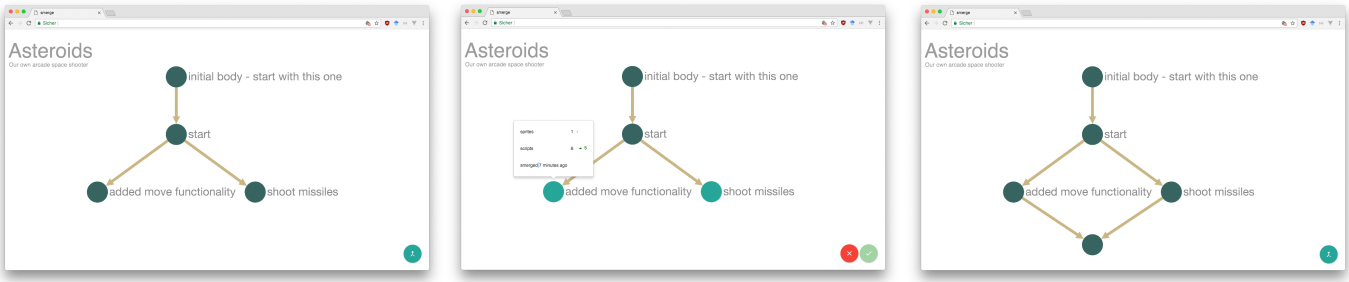


Fig. 2. Merging two versions

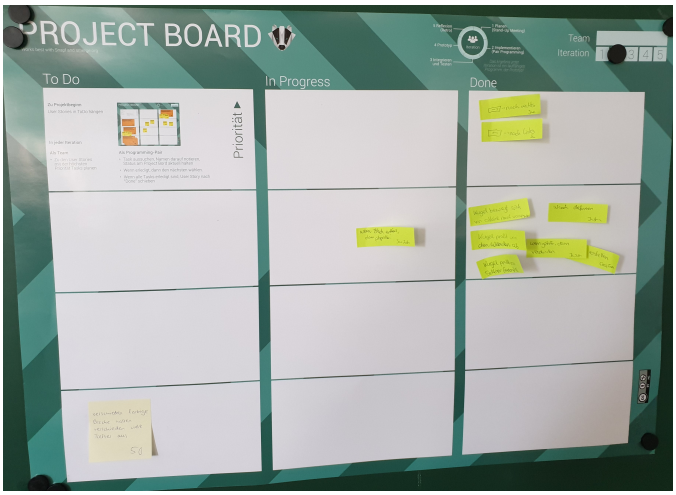


Fig. 3. Example Project board within Iteration 2.

| Pair 1 | | Pair 2 | |
|---|---|---|---|
| **01:38:20** | testing (click green flag) | designing (sprite: projectile) | **1:38:36** |
| **01:39:21** | committing ("shoot with space") | coding (sprite: projectile, script: "when I start as a clone") | **01:38:46** |
| **01:39:40** | - (no action) | testing (click green flag) | **01:39:34** |
| | | coding (sprite: ufo, script: "when I start as a clone") | **01:40:02** |
| | | testing (click green flag) | **01:40:30** |
| | | coding (sprite: projectile, script: "when I start as a clone") | **01:42:39** |
| | | testing (click green flag) | **01:44:46** |
| | | coding (sprite: ufo; script:"When I start as a clone") | **01:44:57** |
| | | testing (click green flag) | **01:45:21** |
| | | committing ("ufos shoot") | **01:45:59** |
| **01:46:12** | closing | merging ("ufos shoot"; "shoot wit space") | **01:46:20** |
| **01:46:30** | exploring | testing (click green flag) | **01:46:46** |
| **01:48:07** | opening (merge of "ufos shoot" and "shoot with space") | | |

Fig. 4. Excerpt from coding table for two programming pairs of one group.

version control system. We also noted down additional information, such as which sprite was edited or what the commit message was. This system has been developed collaboratively up front by both researchers. A similar approach was used by Rosenbaum [24] when evaluating child behavior in musical tinkering software. To avoid neglecting important aspects due to previously-defined categories, inductive additions were allowed. The recorded videos form the basis of the evaluation, while a video segment with a minimum length of 5 seconds serves as a unit of analysis.

In order to allow statements about the usage of the version control system (RQ1), we analyzed the coded events of the individual programming pairs. In a second step, we compared the coded events of all the pairs belonging to a project group (for an example, see figure 4). By considering the pairs' progress and comparing the two pairs of one team, statements could now be made regarding their collaboration (RQ2).

## V. Results

All student groups developed a functioning game. Two of the games were inspired by Asteroids, one was inspired by Breakout and one by Geometry Dash.

### A. RQ1: How Do Novices Use a didactically-adapted Version Control System in a Software Project?

***Finding 1:*** A version control system designed for novices can support students' workflow.

After the short introduction, the students were able to handle all the features. Most of the time could be spent on the actual implementation with Snap! The work within the VCS was goal-oriented and of short duration. The results show, that the concept of a version control system can be simplified in a way that novices use branching and merging.

***Finding 2:*** The graph of the version control system regularly serves as an orientation for the project progress.

After each commit, the students checked the project history graph presented in the version control system to study the groups' progress. Some students closed their version and reopened it, while others just returned to their tab, but all students checked the graph.

***Finding 3:*** Commit messages are not always useful.

While a lot of commit messages reflect which changes had been done (e.g. ”smoother movement” or ”destroyable objects”), there were are also a couple of commits labeled ”version 1:18pm” or ”working prototype ZERO”. This experience matches the literature and our system does not change it either.

***Finding 4**:* No commit before a longer break.

In between the workshop, there was a one hour break. The break was announced to the students well in advance and it was also apparent in the video recordings. However, the groups did not *commit* their latest state to the version control system before that break.

***Finding 5**:* Students include more than one feature per commit, leading to big commits and a long time between them.

The analysis shows that students did not commit based on the features implemented: most of the commits contained more than one new feature or task. In one case, for example, the student pair was adding Game-Over-messages and a handler in case the player touches something, even though the commit was about “destroying objects”. However, every feature committed was functional and can be considered complete.

***Finding 6**:* Students test their code before *committing* and after *merging*.

The students ensured to commit only working versions by testing their code beforehand. Therefore, no commits that made only minor fixes or corrections were coded. The same applies to merging: the students tested the resulting new version and the interaction of the two components.

***Finding 7**:* Student cleanup behavior differs.

The video recordings show that students not only thoroughly test their code, but also that some students clean up the code to increase readability for their peers. They rearrange sprites, remove blocks they used for testing or trying different solutions before they commit their work to the version control system. However, this was not the case for all students. Some also left block fragments in places next to the actual code or positioned the scripts somewhere on the screen.

***Finding 8**:* Students check the versions they wanted to merge. Before completing the merge process, the pair merging the different versions explored all selected nodes to make sure they merge the right versions.

***Finding 9**:* Tinkering on VCS level was barely used.

While the students tinkered a lot within Snap*!*, only one group used tinkering on a VCS level in the sense of using a branch to experiment: After experiencing problems with the framerate, they went back to the earlier version and tried a new approach.

### *B. RQ2: How Do Novices Collaborate when They Work with a didactically-adapted Version Control System?*

***Finding 10**:* All pairs actively managed the VCS.

Activities such as merging were registered in all videos coded. Therefore, there was no clear “project manager” who was in charge of managing the version control system and current project status.

***Finding 11**:* Students tend to split work based on sprites.

The codings show that all groups chose a “topographic” work distribution model, where a sprite was in the focus of each task. Normally, a task included all activities belonging to that sprite, e.g. designing the sprites look and implementing its behavior.

***Finding 12**:* Students avoid conflicts naturally through their design and division of tasks.

During the whole workshop, the students did not experience difficult merge conflicts as they designed and distributed the tasks in a manner that prevented conflicts. Merging mostly meant combining different sprites or different scripts into one project file. We coded no manual conflict resolution. Therefore, all merges were handled by the VCS automatically without the need for manual picking the scripts to keep.

***Finding 13**:* The students aligned their planning to the next merge.

Although the workshop was structured using an agile framework, the students did not align their collaboration process to the given iterations. Thus, they did not only merge at the end of the iteration but before, as well. Sometimes one pair even stopped after finishing a feature and waited for the other pair to finish their task until they could merge – instead of working on other tasks, e.g. implementing further features for the sprite they were working on. The version resulting from the merge was used to plan the further course of action. The students then continued to work on their own or even worked together on the same task. For example, one programming pair implemented the asteroids, the other pair a rocket shooting projectiles. This was merged before the collision detection between asteroid and projectiles was implemented together. Only then, the further procedure was planned and again divided between the two pairs.

## VI. Discussion

This qualitative study aimed to investigate how novices use a didactically-adapted version control system and how they collaborate when they work with it.

### *A. VCS-Usage*

In contrast to the experiences with version control systems outlined in literature, we found novices to use our VCS and its core features intuitively and without hurdles (see finding 1). The students set their project up by themselves without requiring instructors’ help and their interaction with the VCS integrated into the workflow with Snap*!*. Considering the problems for novices with VCS reported in the literature, using a didactically-adapted VCS seems promising.

Nevertheless, when it comes to profound interaction with a VCS, we also found some patterns – also described in the literature – in our data that would benefit from additional instruction.

First of all, the students seem to be unaware of VCS data backup functionality, as they tend to not commit their current progress before a longer break (finding 4). Furthermore, tinkering was mainly done in Snap*!*, without using

the VCS. Products of tinkering that promised little success were removed in Snap! without ever being committed to the VCS. Therefore, the branching feature was mainly used for managing collaboration and not for trying out new ideas. Although a version control system would facilitate tinkering as the storage of all developments is guaranteed, this feature was barely used (finding 9).

However, this behavior could also be attributed to a "holy master" attitude, where only tested and working code is to be committed to the whole project – even if only branches are affected (finding 6, 7 and 8). The rare occurrence of commits (finding 5) supports both of these aspects – the unawareness for the backup functionality as well as a "holy master" attitude. Also the fact that some students left block fragments in places next to the actual code or positioned scripts somewhere on the screen (finding 7) is not a contradiction as this is common among block-based programmers [36].

We also found, once more in accordance with literature, that commit messages are not always named meaningfully (finding 3). In this particular instance, possible reasons for this behavior could be the small overall project size resulting from a one-day workshop as well as the visual graph representation of the project history, which provides additional guidance. Due to these factors, there might have been no need for "more" meaningful commit messages.

As the reported patterns were described in literature (with "professional" VCS) as well as in our study with a didactically-adapted VCS, data backup functionality or meaningful commit messages should be addressed. This could either happen on a tool level or in teaching and instruction. For example, the "holy master"-phenomenon – a reasonable attitude – should not prevent students from tinkering, backing up data, or reverting changes. One possible reaction on a tool level could be the introduction of an additional, separate *graph* representing the releases, which can be selected in the original graph. This way, we create an area explicitly intended for tinkering.

### B. Collaboration and VCS

We were surprised how much the graph and, thus, the VCS led the collaboration process. Our results, as well as unstructured observations from the workshop, showed that the students tended to align their planning more to the VCS and the visual representation of the project status in form of the graph than to the actual project board (finding 2 and 13). Such a visual representation seems vital for a didactically-adapted VCS, as it structures and guides the collaboration.

Furthermore, our findings show that the students "natural" way of collaboration does not incorporate VCS features to its full extent: students collaborate in a conflict-avoiding way (finding 11), for example by splitting tasks sprite-wise (finding 12) and merging often, so that possible conflicts are avoided (finding 13). This might be based on the students existing daily-life experience with (asynchronous) collaboration. For example, when creating a slide show, each student typically works on a different part, as merging their work gets messy otherwise.

While such an approach helps in a lot of situations, using a VCS offers additional options and ways for collaboration: these tools allow for unproblematic merging, therefore tasks could be split not "topographically" but by functionality or the skills of the team members. For example, one student could work on the design of a sprite, while another student implements its behavior. Instead of waiting for other members to finish their tasks (see finding 13), students can continue to work on their tasks and later on include and manage changes in their branch. Therefore, VCS offers different ways of planning and collaboration for projects. Webb recommends to explicitly teach collaboration [37]. Our findings indicate, that ways of collaboration relevant in the context of VCS have to be addressed in teaching, as they are not "natural" for students.

## VII. Conclusion

In this study, we investigated how novices use a version control systems in a block-based environment and how they collaborate when using it. With its focus on asynchronous collaboration and block-based languages, it addresses a scenario that has hardly been explored so far. Although the study was conducted in a one-day workshop setting and the sample size is limited, we can draw three major implications for the use and teaching of version control systems with novices.

First, our results show that novices intuitively use most concepts of version control systems, making them a suitable tool for the high school classroom. Despite having good reasons to use professional VCS even with novices (such as transferability or a "real" look and feel), many of the problems described in literature did not occur with our didactically reduced approach. The integration with the students' usual workflow in block-based languages was smooth and despite the short introduction time, the students were able to work with the tool and the underlying concepts of a VCS immediately.

Nevertheless, the results also show that while the core practices are adopted easily, others are not. For example, the mere use of a didactically reduced version control system does not foster students to tinker or name commits meaningful. Therefore, such patterns need to be addressed within the tools or emphasized in teaching.

Furthermore, our data indicates that the students' "natural" approach to collaboration is conflict-avoiding, e.g. by splitting work sprite-wise and merging as often as possible – even when this results in waiting. The use of version control systems offers unique ways of collaboration, which have to be addressed explicitly in teaching.

In summary, these findings offer deep insights into how novices use a didactically reduced version control system and how they collaborate with such a tool. Building upon this, future research should extend on investigating tool usage over a longer time period. Our study shows that the proposed VCS for block-based languages is easily adaptable for novices and highlights which aspects, especially regarding collaboration,

need to be addressed in teaching. Thus, the findings provide guidance for designing and teaching version control for and with novices.

## References


[1] P. Häkkinen, S. Järvelä, K. Mäkitalo-Siegl, A. Ahonen, P. Näykki, and T. Valtonen, "Preparing teacher-students for twenty-first-century learning practices (prep 21): a framework for enhancing collaborative problem-solving and strategic learning skills," *Teachers and Teaching*, vol. 23, no. 1, pp. 25–41, 2017.

[2] M. M. Lombardi, "Authentic learning for the 21st century: An overview," *Educause learning initiative*, vol. 1, no. 2007, pp. 1–12, 2007.

[3] V. Barr and C. Stephenson, "Bringing computational thinking to k-12: What is involved and what is the role of the computer science education community?" *ACM Inroads*, vol. 2, no. 1, pp. 48–54, Feb. 2011.

[4] J. D. Chase and E. G. Okie, "Combining cooperative learning and peer instruction in introductory computer science," in *Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '00. New York, NY, USA: ACM, 2000, pp. 372–376.

[5] D. Laurillard, "The pedagogical challenges to collaborative technologies," *International Journal of Computer-Supported Collaborative Learning*, vol. 4, no. 1, pp. 5–20, 3 2009.

[6] D. Teague and P. Roe, "Collaborative learning: Towards a solution for novice programmers," in *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ser. ACE '08. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2008, pp. 147–153.

[7] D. Seehorn and L. Clayborn, "Csta k-12 cs standards for all (abstract only)," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: ACM, 2017, pp. 730–730.

[8] Computing at Schools, "Barefoot computing." 2019, https://barefootcas.org.uk/.

[9] L. Haaranen and T. Lehtinen, "Teaching git on the side: Version control system as a course platform," in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '15. New York, NY, USA: ACM, 2015, pp. 87–92.

[10] M. Streeter, "Incorporating real world non-coding features into block ides," in *Proceedings of the 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, ser. BLOCKS AND BEYOND '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 103–104.

[11] T. Michaeli, S. Seegerer, and R. Romeike, "Enabling Collaboration and Tinkering: A Version Control System for Block-based Languages," in *Constructionism 2018: Constructionism, Computational Thinking and Educational Innovation: conference proceedings*, V. D. E. Jasutė, Ed., 2018, pp. 395–403.

[12] J. Roschelle and S. Teasley, "The construction of shared knowledge in collaborative problem solving," in *Computer supported collaborative learning*. Berlin, Heidelberg: Springer, 1995, pp. 69–97.

[13] B. K. Nastasi, D. H. Clements, and M. T. Battista, "Social-cognitive interactions, motivation, and cognitive growth in logo programming and cai problem-solving environments." *Journal of Educational Psychology*, vol. 82, no. 1, p. 150, 1990.

[14] M. Goldman, G. Little, and R. C. Miller, "Collabode: Collaborative coding in the browser," in *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '11. New York, NY, USA: ACM, 2011, pp. 65–68.

[15] K. E. Boyer, A. A. Dwight, R. T. Fondren, M. A. Vouk, and J. C. Lester, "A development environment for distributed synchronous collaborative programming," *SIGCSE Bull.*, vol. 40, no. 3, pp. 158–162, Jun. 2008.

[16] B. Broll and A. Ledeczi, "Distributed programming with netsblox is a snap! (abstract only)," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: ACM, 2017, pp. 640–640.

[17] Y. Ohshima, B. Freudenberg, and D. Amelang, "Kanto: A multi-participant screen-sharing system for etoys, snap!, and gp," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on Programming Experience*, ser. PX/17.2. New York, NY, USA: ACM, 2017, pp. 7–10.

[18] G. Braught, T. Wahls, and L. M. Eby, "The case for pair programming in the computer science classroom," *ACM Transactions on Computing Education (TOCE)*, vol. 11, no. 1, p. 2, 2011.

[19] L. Werner and J. Denning, "Pair programming in middle school," *Journal of Research on Technology in Education*, vol. 42, no. 1, pp. 29–49, 2009.

[20] E. Mendes, L. B. Al-Fakhri, and A. Luxton-Reilly, "Investigating pair-programming in a 2nd-year software development and design computer science course," *SIGCSE Bull.*, vol. 37, no. 3, pp. 296–300, Jun. 2005.

[21] O. Ruvalcaba, L. Werner, and J. Denner, "Observations of pair programming: Variations in collaboration across demographic groups," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16. New York, NY, USA: ACM, 2016, pp. 90–95.

[22] L. Murphy, S. Fitzgerald, B. Hanks, and R. McCauley, "Pair debugging: A transactive discourse analysis," in *Proceedings of the Sixth International Workshop on Computing Education Research*, ser. ICER '10. New York, NY, USA: ACM, 2010, pp. 51–58.

[23] M. Israel, Q. M. Wherfel, S. Shehab, E. A. Ramos, A. Metzger, and G. C. Reese, "Assessing collaborative computing: development of the collaborative-computing observation instrument (c-coi)," *Computer Science Education*, vol. 26, no. 2-3, pp. 208–233, 2016.

[24] E. Rosenbaum, "Explorations in musical tinkering," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2015.

[25] J. Portillo-Rodriguez, A. Vizcaino, M. Piattini, and S. Beecham, "Tools used in global software engineering: A systematic mapping review," *Information and Software Technology*, vol. 54, no. 7, pp. 663 – 685, 2012.

[26] D. M. Case, N. W. Eloe, and J. L. Leopold, "Scaffolding version control into the computer science curriculum," in *Proceedings of the 2016 International Workshop on Distance Education Technology (in conjunction with the 22nd International Conference on Distributed Multimedia Systems (DMS16))*. Salerno, Italy: Knowledge Systems Institute Graduate School, 2016, pp. 175–183.

[27] K. Fisker, D. McCall, M. Kölling, and B. Quig, "Group work support for the bluej ide," *SIGCSE Bull.*, vol. 40, no. 3, pp. 163–168, Jun. 2008.

[28] L. Glassy, "Using version control to observe student software development processes," *J. Comput. Sci. Coll.*, vol. 21, no. 3, pp. 99–106, Feb. 2006.

[29] J. Lawrance, S. Jung, and C. Wiseman, "Git on the cloud in the classroom," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '13. New York, NY, USA: ACM, 2013, pp. 639–644.

[30] P. Brichzin and T. Rau, "Repositories zur Unterstützung von kollaborativen Arbeiten in Softwareprojekt (repositories to support collaborative work in software projects)," in *Lecture Notes in Informatics (LNI): INFOS 2015 - Informatik allgemeinbildend begreifen*. Bonn, Germany: Gesellschaft für Informatik, 2015, pp. 73–82.

[31] K. L. Reid and G. V. Wilson, "Learning by doing: Introducing version control as a way to manage student assignments," *SIGCSE Bull.*, vol. 37, no. 1, pp. 272–276, Feb. 2005.

[32] V. Isomöttönen and M. Cochez, "Challenges and confusions in learning version control with git," in *Information and Communication Technologies in Education, Research, and Industrial Applications*, V. Ermolayev, H. C. Mayr, M. Nikitchenko, A. Spivakovsky, and G. Zholtkevych, Eds. Cham: Springer International Publishing, 2014, pp. 178–193.

[33] R. Romeike and T. Göttel, "Agile projects in high school computing education: Emphasizing a learners' perspective," in *Proceedings of the 7th Workshop in Primary and Secondary Computing Education*, ser. WiPSCE '12. New York, NY, USA: ACM, 2012, pp. 48–57.

[34] J. M. Morse, "Designing funded qualitative research," in *Handbook of qualitative research*, Denzin and Y. S. Lincoln, Eds. Thousand Oaks, CA: Sage Publications, Inc, 1994, pp. 220–235.

[35] P. Mayring, "Qualitative content analysis," *A companion to qualitative research*, vol. 1, pp. 159–176, 2004.

[36] S. Jatzlau, S. Seegerer, and R. Romeike, "The Five Million Piece Puzzle: Finding Answers in 500,000 Snap!-Projects." in *2019 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. IEEE, 2019.

[37] N. M. Webb, "Group collaboration in assessment: Multiple objectives, processes, and outcomes," *Educational Evaluation and Policy Analysis*, vol. 17, no. 2, pp. 239–261, 1995.