Exploring the Benefits and Barriers of Using Computational Notebooks for Collaborative Programming Assignments

Marcel Borowski
Aarhus University
marcel.borowski@cs.au.dk

Johannes Zagermann
University of Konstanz
johannes.zagermann@uni.kn

Clemens N. Klokmose
Aarhus University
clemens@cavi.au.dk

Harald Reiterer
University of Konstanz
harald.reiterer@uni.kn

Roman Rädle
Aarhus University
roman.raedle@cs.au.dk

ABSTRACT

Programming assignments in computer science courses are often processed in pairs or groups of students. While working together, students face several shortcomings in today's software: The lack of real-time collaboration capabilities, the setup time of the development environment, and the use of different devices or operating systems can hamper students when working together on assignments. Text processing platforms like Google Docs solve these problems for the writing process of prose text, and computational notebooks like Google Colaboratory for data analysis tasks. However, none of these platforms allows users to implement interactive applications. We deployed a web-based literate programming system for three months during an introductory course on application development to explore how collaborative programming practices unfold and how the structure of computational notebooks affect the development. During the course, pairs of students solved weekly programming assignments. We analyzed data from weekly questionnaires, three focus groups with students and teaching assistants, and keystroke-level log data to facilitate the understanding of the subtleties of collaborative programming with computational notebooks. Findings reveal that there are distinct collaboration patterns; the preferred collaboration pattern varied between pairs and even varied within pairs over the course of three months. Recognizing these distinct collaboration patterns can help to design future computational notebooks for collaborative programming assignments.

CCS CONCEPTS

• **Human-centered computing** → *Synchronous editors.*

KEYWORDS

computational notebooks; collaborative programming; application development; programming assignments

SIGCSE '20, March 11–14, 2020, Portland, OR, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6793-6/20/03…$15.00
https://doi.org/10.1145/3328778.3366887

ACM Reference Format:
Marcel Borowski, Johannes Zagermann, Clemens N. Klokmose, Harald Reiterer, and Roman Rädle. 2020. Exploring the Benefits and Barriers of Using Computational Notebooks for Collaborative Programming Assignments. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366887

1 INTRODUCTION

Programming assignments in computer science courses often include working on applications or software in teams of multiple students. In our course, in particular, this allows students not only to split up their workload and learn from each other but also provides possibilities for more extensive exercises without increasing teaching assistants' (TA) workload. Although version-control systems like Git allow users to work in asynchronous ways, real-time collaboration as found in platforms like Google Docs [12] is rarely available in development environments. Only recently, code editors like Visual Studio Code [13] and Atom [3] caught up and added functionality for sharing coding environments with others [4, 21].

Code playgrounds (e.g., CodePen [2] or JSFiddle [8]) are platforms where web developers can "explore and test" code without having to set up a development environment. Code playgrounds allow developers to tinker with code in the browser, quickly build prototypes, and share these with others. However, the application state of these prototypes is ephemeral, meaning that it does not persist beyond page reloads, which makes code playgrounds primarily a tool for exploration and testing. Building functional applications still requires developers to transfer code from a playground into their software development tool. In recent years, also the computational notebook has become an increasingly popular tool, particularly among the data science community. Examples like Jupyter Notebook [9], Google Colaboratory [11] and Observable [16] allow users to mix text and executable code in the same document following the *literate computing* paradigm [14]. A literate computing environment "can be seen as a blend of a command-line environment such as Unix shell with a word processor" [14].

Tools that combine the approaches of real-time collaboration documents, code playgrounds, and computational notebooks are still in their infancy (i.e., combining exploration and testing of code with the possibility of building functional applications [16, 20]); and their benefits to users are under-explored. A deeper understanding of how these tools would affect the collaborative development of interactive systems is yet lacking—both in the classroom and among professional programmers. In this paper, we report on our experiences of deploying an open-source collaborative computational notebook [20] for three months during an introductory university course on application development in human-computer interaction (HCI) to explore how students collaborated on the development of interactive systems. As part of the course assignments, pairs of students were tasked to develop eight small interactive applications based on the textbook *Designing Interfaces* [25].

Our experiences show that the linear structure of notebooks can be beneficial, especially when catching up on past activities or when navigating code. Real-time collaboration, the "no setup" time, and the shareability of notebooks were valued by students and TAs. Behavioral analysis revealed distinct collaboration patterns. The preferred collaboration style varied between pairs and even varied within pairs over the course of the three months. However, we also identified several breakdowns, for example, during real-time collaborative development when pairs worked in the same document. Moreover, we identified several tool requirements for computational notebooks used by students in a collaborative fashion (e.g., code completion). Recognizing the distinct collaboration styles and tool requirements can help in the design of future collaborative literate computing tools and in the assessment of the suitability of a tool for collaborative programming assignments.

In the remainder of this paper, we present related work, our tool for assignment facilitation, an overview of the course as well as our analysis and findings. Finally, we discuss the benefits and barriers of using computational notebooks for programming assignments.

2 RELATED WORK

Jupyter Notebook [9] is a popular implementation of computational notebooks supporting the literate computing paradigm. It allows users to create cells for narrative text (including rich-text editing) and code (e.g., Python). Code cells can be used to generate text output or visualizations. Observable [16] omits the separation of text and code and allows users to create *reactive* cells—i.e., dependent cells automatically update or execute—that can contain both text and executable code. The front end of both, Jupyter and Observable, can be accessed with a web browser. Google Colaboratory [11] is a platform that is also accessible with a web browser, supporting real-time collaboration similar to Google Docs [12].

Codestrates [20] is a web-based platform that enables users to develop applications or create documents with embedded computation resembling a notebook-like environment—i.e., *codestrates*. A codestrate is structured in sections and paragraphs. It offers body, code, style, and data paragraphs. A body paragraph can be used to create the structure of a user interface (using HTML), a style paragraph can be used to style the content of a body paragraph (using CSS), a code paragraph can be used to implement interactivity (using JavaScript), and a data paragraph can store arbitrary data (using JSON). Body paragraphs can be set to full-screen, effectively turning them into web apps. Sections can be used to group paragraphs and structure the document. Like Colaboratory, Codestrates allows for real-time collaboration—not only for editing code of computations, but also for reprogramming and extending the notebook itself.

The use of computational notebooks for teaching or learning has been explored in areas like artificial intelligence or chemistry. O'Hara et al. exploited Jupyter to teach artificial intelligence in "easy-to-use interfaces" [17]. Srnec et al. [22] explained reciprocal space to undergraduates by providing a notebook that converts real space vectors into reciprocal space vectors. Both valued the possibility to combine theory and materials from a lecture with code and executable equations. Other authors looked into the teaching of computing [27] and how notebooks could be used to create and "autograde" assignments [6]. The usage of notebooks for collaborative application development is still mostly under-explored.

The collaboration of groups of students that process programming assignments in an educational context has been frequently analyzed for the practice of pair programming. In their extensive meta-analysis, Umapathy and Ritzhaupt find "that pair programming, if implemented properly, can have a positive impact on students' programming assignment grades, exam scores, and persistence in computer programming courses" [26]. While past research mostly focused explicitly on pair programming, where two students have specific roles—the *driver* who writes code and the *navigator* who reviews the written code—our work explores how collaboration unfolds in an unconstrained symmetric collaboration setting, where students can decide how they collaborate or split up the workload.

3 A TOOL FOR ASSIGNMENT FACILITATION

We deployed a literate computing platform in an introductory course on HCI to get insights on its use during the implementation of interactive applications. The course was split into two parts. Part one were weekly lectures covering basic topics of HCI where students learn about how to design *usable* interactive systems. For example, topics that were covered were human visual perception and design principles such as affordances and constraints [15]. Part two was the practical part with weekly tutorials and assignments; both accompanied the lectures. In the tutorials, TAs presented and discussed different user interface design patterns [25] with students. For the assignments, pairs of students developed small interactive prototypes based on these patterns and handed them in weekly. Assignments had to be passed to proceed to the next assignment.

3.1 Enabling Symmetric Collaboration

In the previous years of the course, assignments were based on web technologies and used JavaScript as their programming language. Thus, we chose to use Codestrates [20] as a computational notebook platform as it allows to develop stateful applications using web technologies, which—in contrast to other open-source implementations such as Jupyter [9] or Observable [16]—inherently supports real-time collaboration similar to a Google Docs document. We adapted Codestrates to operationalize our research by building small extensions to it using Codestrates' package management [1]. The package management allows users to share content as *packages* among codestrates. Packages can be *pulled* from another codestrate or *pushed* into one. This allowed us to use dedicated codestrates as shared *repositories* to distribute the assignments to students, and to provide repositories to collect completed assignments.

3.2 Assignments and Course Material

The assignments focused on the learning of interactive systems design. Students had to implement eight small applications using different interface design patterns derived from the textbook *Designing Interfaces* [25]. For example, assignment three required them to implement a to-do list as a combination of the design patterns *Row Striping* and *Input Prompt*. In Codestrates, the assignments had six paragraphs, which were formatted in the following structure:

(1) A task description, requirements, and the deadline.
(2) Four paragraphs (i.e., preview, body, style and code) served as a development environment.
(3) A questionnaire about their use of the notebook.

Students needed to use a body paragraph for their HTML, a style paragraph for their CSS styles, and a code paragraph to add their JavaScript code. Upon running the code paragraph, the content of the body HTML was loaded into the preview paragraph and the code of the students was executed. The preview paragraph rendered the body, style, and code paragraphs into a working application—similar to classic web development.

3.3 Course Management

Similarly to the assignments, the tutorial was managed using Codestrates. Students were provided with an overview course website including links to important repositories such as assignments, sample solutions of past assignments, and frequently asked questions. The TAs used dedicated repositories to handle to-dos, score tables, and group-related data (i.e., contact data of group members). Additionally, TAs created slide decks for each tutorial in our system. The web-based and dynamic nature of the platform allowed for easily sharing links across TAs for further organizational activities, directing support for students, and distributing of tutorial materials.

3.4 Workflow

The TAs prepared every assignment in a *private* codestrate that was only accessible to them. After each weekly tutorial, they pushed the assignments into a *public* repository that was accessible to students. The students then created a new codestrate using a link on the course website or used a previous codestrate, pulled the assignments into it, worked on the assignment, and pushed it to their dedicated submission repository. They were able to create any number of codestrates to work on and could push their submissions as many times as desired. The TAs finally pulled the submitted assignments into their *private* codestrates, corrected them and pushed the corrected versions back to the pairs' repositories (see Figure 1).

Similarly to the assignments, TAs prepared slide decks and example solutions for each tutorial in *private* codestrates and pushed them after the tutorial session to the *public* repository. This allowed students to pull, e.g., slides and example solutions into their codestrates for comparison or further reading.

4 COURSE OVERVIEW

4.1 Participants

58 students participated in the course that was organized and conducted by three TAs. Students worked in pairs on assignments, forming 29 pairs. Two TAs had previous experience with teaching the course while one TA had just passed the course the year before.

Within the first three weeks, six of the 29 pairs dropped out of the course. The stated reasons were that they either did not have enough time for the course or decided to take the course in the following year. These six pairs were excluded from the data analysis. Of the remaining 46 students (23 pairs), 30 students were in their fourth semester, four students were in their third semester, four were in their sixth semester, one student was in an "other" semester, and five students did not disclose this information. 17 students and two TAs were male, three students and one TA were female, and 26 students did not disclose their gender.

Students reported about their prior programming experience on a five-point Likert scale ranging from *Beginner* (1) to *Expert* (5): two reported to be experts (5), four reported some experience (3), 11 declared to be rather beginners (2), and three reported to be complete beginners (1). 11 students reported to have used JavaScript before. 26 students did not disclose their prior programming experience.

We recommended using Google Chrome [10] because Codestrates was developed and tested in that browser. Students could to solve the assignments in any way that suited them, allowing for different styles of collaboration (e.g., co-located or remote).

4.2 Course Procedure

The first three tutorial sessions included: (i) an introduction of topics, procedure, and pairing of students, (ii) an introduction and recap of web technologies such as HTML, CSS, and JavaScript, and (iii) an introduction to how the system is used for the course management and assignments. These sessions provided practical step-by-step guides on how to retrieve, process, and submit assignments. They also compensated for differences in students' prior knowledge and served as a training phase for the computational notebook system.

Following this, eight weekly assignments were provided. New assignments were briefly presented and handed out at the end of weekly tutorials. The students then had one week to push their solution. The tutorial session following the submission of an assignment was used to discuss sample solutions, challenges, and obstacles in previous assignments. Students were free to ask for support from the TAs (e.g., to solve minor bugs) during the semester.

4.3 Data Analysis

At the end of each assignment, we asked students to fill out a questionnaire regarding their system usage, working style, problems, and suggestions for improvement. At the end of the course, we conducted a focus group with four students—each from a different pair. We also interviewed each TA on a weekly basis regarding their system usage, working style, and problems. After the weeks four and eight, we conducted two focus groups with all three TAs.

During the entirety of the course, all package pushes and pulls were logged, including information about the packages, the user who performed the action, the codestrate ID that pushed or pulled the package, and the repository. This allowed us to trace codestrates that participants created during the course. For each codestrate, the server recorded the edit history—down to the keystroke-level. Each record contains information such as the user ID, the client's network IP address, and a timestamp indicating when the edit was made. All edits by the same user that were performed fewer than 20 minutes apart were combined into *sessions*. We used empirical data sampling to find a threshold for the duration of a session—20 minutes proved

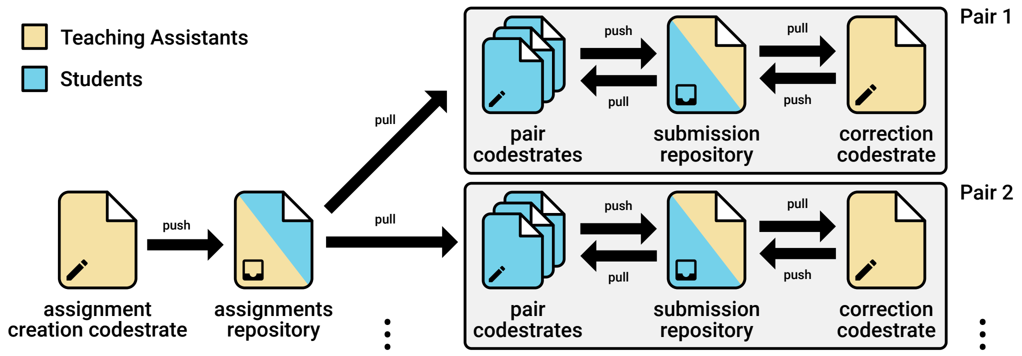

Figure 1: The workflow of assignments. TAs create assignments and push them to a repository available to students. The students pull the assignments, process them in one or multiple codestrates and submit them to their submission repository. TAs pull the submissions, correct them and push them back to the submission repository.

to be a good fit. Sessions were marked as *collaborative* whenever the sessions of each partner on the same codestrate overlapped.

To analyze the working styles of pairs, we analyzed each assignment resulting in $23 \cdot 8 = 184$ *submissions*. Within each submission, we refer to students as S1 and S2. We manually coded each submission for six attributes and used them to assess working styles:

(1) *Users:* Did S1, S2, or both work on the assignment?
(2) *Time:* Did students work at the same time or at different times on the assignment?
(3) *Location:* Did students work co-located or remotely?
(4) *Workflow:* Did students process the assignment all at once or in multiple sessions?
(5) *Documents:* Did students use one or multiple codestrates?
(6) *Work Split:* Did students work on every part of the assignment together or did they split up the parts of the assignment (e.g., the structure of the to-do list and its interactivity)?

The attributes *Users*, *Workflow*, and *Documents* were derived from the log data, *Work Split* was derived from the questionnaire, and *Time* and *Location* were derived and matched from both.

5 FINDINGS AND DISCUSSION

We report on our findings of collaborative working styles and the structure of computational notebooks. The pairs of students will be referred to as P1 to P23, the participants of the focus group with students as F1 to F4, and the TAs as T1 to T3.

5.1 Collaborative Working Styles

To identify distinct working styles, we analyzed each submission according to their attributes. We also investigated questionnaire data to reveal other influences on the collaboration between students.

5.1.1 Finding 1.1: Collaboration Patterns. The identified collaboration patterns were grouped in synchronous and asynchronous collaboration and a pattern where only one student worked in the computational notebook.

Synchronous collaboration: Pairs of students worked together synchronously on 48 of 184 submissions. Of those 48 submissions, collaboration was co-located in 19, and remote in 29 of them (see Figure 2, rows 1 and 2). Most of the time (40 submissions), the pairs worked together on all parts of the assignment, whereas in only 8 submissions students divided and conquered when working together at the same time. Splitting work was, e.g., done by working on different paragraphs (e.g., HTML, CSS, and JavaScript) or by dividing assignments into different topics (e.g., one student creating an online form and the other adding the validation for the form).

Two pairs used synchronous co-located collaboration as their primary working style (P6: 6 submissions; P18: 5 submissions) and three pairs used synchronous remote collaboration (P20: 7 submissions; P10, P13: 6 submissions).

Asynchronous collaboration: In 75 of 184 submissions, students worked together asynchronously. When collaborating in this way, pairs worked remotely in all instances. Students leveraged three different ways to share intermediate results with their partner: (i) both students visited the same notebook (see Figure 2, row 3); (ii) pairs used their submission repository as a shared repository; (iii) students shared code by copying it across notebooks.

Nine pairs used asynchronous collaboration as their primary working style (P2, P5, P12: 8 submissions; P11: 7 submissions; P1, P3, P19: 6 submissions; P7, P21: 5 submissions).

Only one student works in the computational notebook: In 61 of the 184 submissions, only one student worked during the assignment in the computational notebook, i.e., no edits from the second student appeared in the log (see Figure 2, row 4). However, in 23 submissions, pairs reported that they did the assignment together but did not split work. This indicates that both students could be sitting together in front of the same device or collaborating remotely using screen sharing. Triangulating data from questionnaires and edit logs helped to uncover behavior that is otherwise obscured, namely, who is sitting in front of the device. In six pairs, only one person worked in the computational notebook for the majority of the time (P8, P14, P15, P22: 8 submissions; P4, P9: 5 submissions).

5.1.2 Finding 1.2: Benefits and Barriers of Real-Time Synchronization. Students reported that they valued how edits in their notebooks are synchronized instantly, as it made it easier for them to collaborate. Our results support this by showing that pairs used only a single notebook per assignment in 50 of the 123 submissions, in which both students contributed in the notebook.

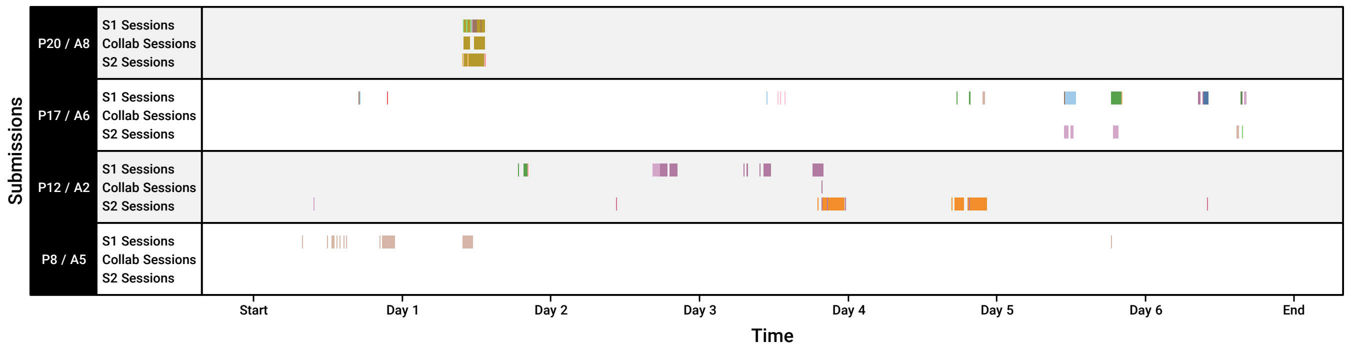

Figure 2: Examples of collaborative working styles. Each row shows one exemplary submission (e.g., pair 20 and assignment 8 in the first row). Blocks indicate sessions for S1, S2, and collaborative sessions—colors indicate the use of different documents.

Compared to version-control systems like Git, students liked that there were no conflicts, as changes were immediately applied for each partner. However, students wished for a better change history, for example, seeing what their partner did since the last time they visited the notebook.

> *When I, for example, edit something in a project and my partner goes online . . . [I wish] that he is notified of what I changed most recently.* (F3)

Students also reported downsides to synchronized notebooks. For example, they occasionally had problems with jumping code (due to another person writing above) or accidentally overwriting their partners' code. Log data shows that, for instance, P17 moved from using a shared notebook in the first five assignments to using an individual notebook per partner for the last three assignments.

5.1.3 Finding 1.3: Shareability via Links Supports Collaboration. Both the students and all TAs reported that sharing documents via links made assignment correction and supporting students easier. For example, in previous years TAs had to download zipped project archives with multiple files from a learning management platform provided by the university and run the projects locally on their machine. Incompatible runtime environments led to solutions not working correctly. The TAs reported that supporting students was also easier compared to the previous year, as the students just needed to provide the link to their notebook in order for TAs to help them, as TAs had access to all codestrates.

Also, students reported in the questionnaires and the focus group that they see value in the shareability—especially when sharing documents with people with a non-technical background.

> *If you want to quickly send something to someone who is unfamiliar with web development, you can just send a link instead of all the project files.* (P2)

5.1.4 Finding 1.4: Submission Repository as a Fail Safe. Three of the four students in the focus group reported that they would use their submission repository as a way to back up their current state of work. They pushed assignments multiple times per week to their submission repository. This was also confirmed by the logging data.

> *We mainly used it [pushing our solution to the submission repository] as a checkpoint. Especially when integrating new functions . . .* (F3)

Students reported that they pushed their current solution to their submission repository as a "checkpoint" (F3) or fail safe when trying out new code that could possibly destroy their notebook (e.g., by an endless loop). However, not every pair worked like this; F1 and their partner just pushed the completed assignment once at the end of each week and finished assignments together in one go.

5.1.5 Discussion. Our findings show that pairs adopted various working styles to solve the assignments. While many pairs collaborated synchronously, asynchronous collaboration was the most common working style among pairs. This indicates that not only synchronous (cf. [28]), but also asynchronous collaboration benefits from real-time synchronization and easy shareability. The asynchronous collaboration is similar to [23], where students could help each other on writing assignments at any time, allowing collaboration "without restriction of time and space" [23]. Additionally, students used the computational notebook not only to solve the assignment but also for coordination—similar to the results of Olson et al. [18]—and handover of different states either via a shared notebook or the submission repository. However, collaboration capabilities have room for improvement: students missed tools to keep track of their partners' work when working asynchronously—although changes are logged by the server, they are not accessible in a convenient way—and encountered the problem of jumping content when working synchronously. This demand for knowing what and when their partner was working is in line with the elements of workspace awareness by Gutwin and Greenberg [5].

There is a trade-off in synchronizing the notebook content of two users: synchronizing content at a keystroke-level can hinder individual work, such as when contents in the notebook jump, but it is difficult to coordinate and maintain awareness when there is not enough contextual information about one's partner; features like remote cursors or pointers are important.

While the contextual information discussed above is useful for synchronous collaboration, asynchronous collaboration—the most frequent collaboration pattern—is often overlooked in terms of providing similar types of tools. Future computational notebooks should focus on supporting asynchronous collaboration. Being able to keep track of the changes that other users performed could ease splitting up work and putting results together. Collaborative tools from other platforms such as Overleaf [19] or Google Docs could further support asynchronous collaboration, e.g., commenting, highlighting tracked changes, or providing an edit history.

To address the issue of jumping content when working synchronously, future computational notebooks could provide different modes for different working styles. For instance, when dividing and conquering work, one mode could turn off synchronization for the user interface or allow the pinning of paragraphs. When working on the same parts of an application, a more tightly-coupled mode could allow both users to see cursors and pointers of their partners to create an awareness of what the other is working on. This future work could also be informed by previous research on collaborative sensemaking on interactive tabletops [7, 24].

5.2 Web-based Computational Notebooks

Computational notebooks differ from classic file-based development environments: Instead of files and folders, users structure their programs in sections and paragraphs. We investigate differences to better understand their suitability as a development environment.

5.2.1 Finding 2.1: "No Setup" Time Assists Initial Phases of Development. Both students and TAs reported that they liked how they could directly focus on the content of the assignments, in contrast to setting up the project in a development environment. As the computational notebook runs in the web browser, it works across platforms. All students participating in the focus group reported that the easy setup helped them to focus on solving the assignments.

> *It takes the work out of your hands. It lets you concentrate on all of the important things. Implementation etc. How to realize things.* (F3)

They did not have to find out how to add libraries to their projects, e.g., downloading files and adding them to their projects. They could add a library through a dialog from within the notebook. TAs also noted how quick the process of getting to the actual work was. T2 described it as "plug-and-play" where "you can just open a new codestrate or an existing project and it would directly run, unlike projects in folders that need a development environment."

5.2.2 Finding 2.2: Linear Structure Assists Students. The linear structure of the notebooks supported students in solving assignments. It encouraged them to focus on one part at a time: First writing the body of the application with HTML, then styling it with CSS, and lastly adding interactivity through JavaScript. This also helped them to split up work with their partner. For instance, one student wrote the HTML and JavaScript while the other styled it via CSS.

> *I also like it very much, because I have this division into sub-problems. From HTML to CSS to JavaScript.* (F2)

In contrast, T2 and T3 reported that the linear structure restricted the customizability of their notebooks. Not being able to watch two paragraphs side by side within a notebook forced them to open the same notebook in multiple tabs or only working in one paragraph at a time. Similarly, F2 reported that the fixed-size preview paragraph in the assignments hindered developing a responsive interface.

5.2.3 Discussion. The lower threshold to set up and use a computational notebook allowed participants—both students and TAs—to quickly get started and tinker with code. They found themselves comfortable in using sections and paragraphs instead of files. The linear structure supported students' workflow, as the visual separation of paragraphs guided them to split up the assignment into subtasks. This is similar to how computational notebooks for data analysis like Jupyter [9] split the analysis into cells.

Although our computational notebook supports novice users in initial phases, the linear structure of computational notebooks can also be a limitation for more experienced users. For some tasks such as the comparison of different paragraphs, this resulted in additional scrolling activities that hampered the ability to compare paragraphs. It would be an improvement for users to be able to break up the linear structure, e.g., showing two paragraphs side-by-side.

6 CHALLENGES

The analysis of the log data revealed students' working styles where the majority of students worked mostly at different places and at different times. The collaborative notebook allowed them to combine or merge their individual solutions by, for instance, visiting the same notebook, using their submission repository, or by copying and sending parts of their code to their partner. However, the log data does not provide insights into interpersonal communication—such as face-to-face or via messaging apps—that was used to coordinate activities. The log data showed several instances where only one student was actively working on the assignment in a notebook. This does not necessarily mean that these pairs did not collaborate; as the questionnaires showed, some pairs worked at the same time and place during these assignments (e.g., using the same laptop together). Additionally, we cannot exclude other possibilities, like one student working on a local code editor or code playground while the other one combines their solution in the notebook.

7 CONCLUSION

We successfully deployed a collaborative computational notebook in an introductory course on application development in HCI to investigate the collaborative working styles of students. Combining the data from questionnaires, focus groups, and keystroke-level log data allowed us to get insights into the distinct collaboration patterns of student pairs when solving assignments and implementing small interactive applications. Our findings highlight the importance of a development environment that can support multiple working styles; as students incorporate diverse working styles ranging from synchronous, asynchronous, to apparently no collaboration at all, as well as from co-located to remote settings.

The experiences in our course, further, highlight that collaborative web-based platforms benefit the collaboration of students, the communication with TAs, and give students a head start by working "plug-and-play" in the web browser without the need for setting up an environment. Dividing assignments into logical chunks, like in the linear paragraph structure in computational notebooks, made it easier for beginners to get started with their assignments.

We believe, this report will be of value for computer science educators that intend to incorporate collaboration into their assignments. By providing insights into different styles of collaboration, we hope our work also motivates other educators to think more about the collaborative aspects when designing assignments.

ACKNOWLEDGMENTS

We thank Lindsay Reynolds for proof-reading this paper and Bjarke Vognstrup Fog for valuable feedback. This research was funded by the *Deutsche Forschungsgemeinschaft* (DFG, German Research Foundation) — Project-ID 251654672 — TRR 161 (Project C01), and the *Aarhus University Research Foundation.*

REFERENCES

[1] Marcel Borowski, Roman Rädle, and Clemens Nylandsted Klokmose. 2018. Codestrate Packages: An Alternative to “One-Size-Fits-All” Software. *CHI EA ’18 Proceedings of the 2018 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (2018). https://doi.org/10.1145/3170427.3188563

[2] CodePen. 2019. *CodePen.* https://codepen.io Last accessed: November 11, 2019.

[3] GitHub, Inc. 2019. *Atom.* https://atom.io Last accessed: November 11, 2019.

[4] GitHub, Inc. 2019. *Teletype for Atom.* https://teletype.atom.io/ Last accessed: November 11, 2019.

[5] Carl Gutwin and Saul Greenberg. 2002. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work (CSCW)* 11, 3 (01 Sep 2002), 411–446. https://doi.org/10.1023/A:1021271517844

[6] Jessica B. Hamrick. 2016. Creating and Grading IPython/Jupyter Notebook Assignments with NbGrader. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE ’16).* ACM, New York, NY, USA, 242–242. https://doi.org/10.1145/2839509.2850507

[7] Petra Isenberg, Danyel Fisher, Sharoda A. Paul, Meredith Ringel Morris, Kori Inkpen, and Mary Czerwinski. 2012. Co-Located Collaborative Visual Analytics Around a Tabletop Display. *IEEE Transactions on Visualization and Computer Graphics* 18, 5 (May 2012), 689–702. https://doi.org/10.1109/TVCG.2011.287

[8] JSFiddle. 2019. *JSFiddle.* https://jsfiddle.net Last accessed: November 11, 2019.

[9] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter Notebooks — a publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), 87–90. https://doi.org/10.3233/978-1-61499-649-1-87

[10] Google LLC. 2019. *Google Chrome.* https://www.google.com/chrome Last accessed: November 11, 2019.

[11] Google LLC. 2019. *Google Colaboratory.* https://colab.research.google.com Last accessed: November 11, 2019.

[12] Google LLC. 2019. *Google Docs.* https://docs.google.com Last accessed: November 11, 2019.

[13] Microsoft. 2019. *Visual Studio Code.* https://code.visualstudio.com Last accessed: November 11, 2019.

[14] K. Jarrod Millman, Fernando Pérez, Victoria Stodden, Friedrich Leisch, and Roger D. Peng. 2014. Developing open source scientific practice. *Implementing Reproducible Research* 149 (2014).

[15] Donald Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition.* Basic Books.

[16] Observable, Inc. 2019. *Observable.* https://observablehq.com Last accessed: November 11, 2019.

[17] Keith J. O’Hara, Doug Blank, and James Marshall. 2015. Computational Notebooks for AI Education. (2015). https://doi.org/10.13140/2.1.2434.5928

[18] Judith S. Olson, Dakuo Wang, Gary M. Olson, and Jingwen Zhang. 2017. How People Write Together Now: Beginning the Investigation with Advanced Undergraduates in a Project Course. *ACM Trans. Comput.-Hum. Interact.* 24, 1, Article 4 (March 2017), 40 pages. https://doi.org/10.1145/3038919

[19] Overleaf. 2019. *Overleaf.* https://www.overleaf.com Last accessed: November 11, 2019.

[20] Roman Rädle, Midas Nouwens, Kristian Antonsen, James R. Eagan, and Clemens N. Klokmose. 2017. Codestrates: Literate Computing with Webstrates. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST ’17).* ACM, New York, NY, USA, 715–725. https://doi.org/10.1145/3126594.3126642

[21] Amanda Silver. 2017. *Introducing Visual Studio Live Share.* https://code.visualstudio.com/blogs/2017/11/15/live-share Last accessed: November 11, 2019.

[22] Matthew N. Srnec, Shiv Upadhyay, and Jeffry D. Madura. 2016. Teaching Reciprocal Space to Undergraduates via Theory and Code Components of an IPython Notebook. *Journal of Chemical Education* 93, 12 (2016), 2106–2109. https://doi.org/10.1021/acs.jchemed.6b00392

[23] Ornprapat Suwantarathip and Saovapa Wichadee. 2014. The Effects of Collaborative Writing Activity Using Google Docs on Students’ Writing Abilities. *Turkish Online Journal of Educational Technology - TOJET* 13, 2005 (2014), 148–156. https://eric.ed.gov/?id=EJ1022935

[24] Anthony Tang, Melanie Tory, Barry Po, Petra Neumann, and Sheelagh Carpendale. 2006. Collaborative Coupling over Tabletop Displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI ’06).* ACM, New York, NY, USA, 1181–1190. https://doi.org/10.1145/1124772.1124950

[25] Jenifer Tidwell. 2010. *Designing Interfaces.* O’Reilly Media, Inc.

[26] Karthikeyan Umapathy and Albert D. Ritzhaupt. 2017. A Meta-Analysis of Pair-Programming in Computer Programming Courses: Implications for Educational Practice. *ACM Trans. Comput. Educ.* 17, 4, Article 16 (Aug. 2017), 13 pages. https://doi.org/10.1145/2996201

[27] Greg Wilson, Fernando Perez, and Peter Norvig. 2014. Teaching Computing with the IPython Notebook (Abstract Only). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE ’14).* ACM, New York, NY, USA, 740–740. https://doi.org/10.1145/2538862.2539011

[28] Soobin Yim, Dakuo Wang, Judith Olson, Viet Vu, and Mark Warschauer. 2017. Synchronous Collaborative Writing in the Classroom: Undergraduates’ Collaboration Practices and Their Impact on Writing Style, Quality, and Quantity. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW ’17).* ACM, New York, NY, USA, 468–479. https://doi.org/10.1145/2998181.2998356