Synchronous, In-Context Peer Support for Rovercode

Brady Hurlburt
Georgia Institute of Technology
Atlanta, Georgia
brady.hurlburt@gatech.edu

Bryce Montano
Georgia Institute of Technology
Atlanta, Georgia
hellobryce@gatech.edu

ABSTRACT

UPDATED—December 9, 2018. Asynchronous support platforms like help forums are common in the field of software engineering, but are they the most effective tool for facilitating peer support among students who are learning to code? This paper proposes an online peer-support system that connects students through synchronous communication with rich problem context. First, the peer-support practices of existing sites like Scratch and Glitch are examined through the lense of constructionism and social learning theory. Then, these lessons are applied to the design of a peer-support platform for the open-source classroom robotics platform, Rovercode.

ACM Classification Keywords

K.3.1. Computers and Education : Computer Uses in Education

Author Keywords

online peer learning; social learning; constructionism; Scratch; Glitch

INTRODUCTION

Asynchronous support platforms like Stack Overflow are prominent in the field of software development. On these platforms, users post a carefully posed question, then wait for responses that could come at any time. Because these forums strive to be encyclopedic, posters are often encouraged to remove as much context from their question as possible, leaving the most generally applicable form of their problem [20].

In stark contrast to these forums, real-time support systems aim to show that learning can flourish with ephemeral, synchronous interactions between peer learners. These platforms facilitate peer support that is synchronous and tightly tethered to the context of the learner's problem. Are platforms like these promising for student coding environments? Are they scalable in online communities? Do they encourage students to cheat? Are they safe for children?

This paper answers these questions and provides a justification for such platforms based on constructionism and social learning theory Examples from existing sites like Scratch and Glitch are used to show that these support platforms encourage the quick proliferation of knowledge through their communities and create socially gratifying experiences for students. Finally, the lessons from this investigation are applied to the design of a peer-support platform for the open-source classroom robotics platform Rovercode.

PREVIOUS WORK

Constructionism and Social Learning

In Papert's constructionism, the people around the student supply the models and metaphors to use as raw materials for building understanding [14]. Bandura's social learning theory corroborates that idea: people can learn by observing and interacting with those around them [19] and with the learning artifacts created by those around them [9].

The sections below examine how these principals can be used in peer-support systems.

Why Synchronous? Examining ReachOut and Glitch

Researchers Ribak, Jacovi, and Soroka built a real-time chat-based support platform called ReachOut and studied its efficacy [17]. They found that synchronous support sessions were effective for quickly transferring understanding of the full context of the question from the person requesting support to the support provider, a process they call "negotiation of meaning." Specific details about the context of the question are sometimes undesired in encyclopedic forums like Stack Overflow [20], but context richness was embraced by ReachOut.

The ReachOut experiment questioned the oft-adopted motto "Search before you ask." It found that even when institutional knowledge was documented and searchable, chatting with peers was often preferable because a peer could quickly summarize information and help fit the information into one's existing understanding (a key activity in constructionism). Furthermore, participants contributing to shared effort felt a social gratification that asynchronous forums did not provide [17].

The most prominent modern implementation of ReachOut's ideas is an online platform called Glitch [8]. Glitch is an online community and toolchain for creating and deploying web applications. They offer a "raise your hand" feature which allows users to request help directly from the code editor (refer to Figure 1). The support request is displayed to other users on the Glitch home page. If someone volunteers to help, she is connected to a live collaborative editing session with the support requester. In the session, the support provider can add and edit code and leave explanatory comments. When the

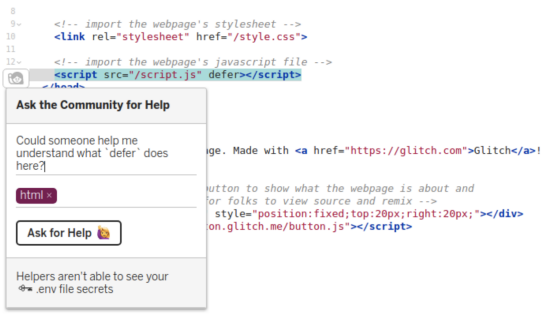


Figure 1: Glitch's "raise your hand" feature [8].

question is resolved, the requester ends the session and has an opportunity to send a "Thanks!" message to the provider.

Why In-Context? Examining Scratch

Social learning theory calls for students to interact with artifacts of each others' work, and MIT's Scratch platform is a prominent example of an online community that facilitates this. Scratch is a children's programming and community platform created by the MIT Media Lab's Lifelong Kindergarten group [16]. Scratch hosts millions of users who create content through graphical coding. Users can share their work publicly and allow others to "remix" (i.e. to clone and edit their work). Remixing places users in direct contact with each others' work products, which is critically different than interacting with an isolated question on a forum. Remixing allows skills and knowledge to spread quickly [7], and has been proven on Scratch and other platforms to increase the quality of the community's creative output as a whole [13] [2].

What About Cheating?

Collaboration or Plagiarism?

If a student's behavior in a peer-support activity resembles copying, is the student learning? To answer this question, it is important to keep in mind that the goals of a peer-learning activity may differ from those of a traditional learning activity. A copying behavior that would land squarely within the definition of plagiarism in a language-arts classroom may be precisely the adaption-by-observation sought in a peer-learning activity [4]. Learning must be assessed after peer-support activities, but the assessment must look beyond isolated achievement and include self-assessment, peer-assessment, assessment of process, and negotiated assessment [3].

Learning or Freeloading?

If a student's behavior in a peer-support activity makes it appear that she simply wants someone else to do their work for her, is the student learning? And can the peer-support community handle the burden of those questions? This question was brought to the popular consciousness of the computer programming field by a weblog post by Joel Spolsky of Stack Overflow [20].

Studies in the Scratch community have shown that the freeloading attitude can be prevented. Researcher Champika Fernando showed that placing members in roles focused on mentorship and teaching can help shift their focus from question-asking to learning [6].

Is it Safe?

A Focus on Sharing Work Products Reduces Abuse

Many peer-support systems take place online, where is critical to guard the safety and privacy of children who participate. However, misapplied moderation strategies can be detrimental to the community. Constraints to narrow the expressiveness of communication are easily circumvented [15]. Off-loading the entire task of conflict resolution to adults leaves students helpless to navigate even the slightest tension with their peers [18]. Down-votes on contributions by new members risk pushing the member away while they are still learning community norms [12], and worse yet, early down-votes counter-intuitively encourage a user to post more low-quality content [5].

The burden on moderation, however, is lighter in creator communities like Scratch, which reports a relatively low rate of abusive communication. This is likely because the interactions in Scratch are not simply social, but rather have a purpose – to share work products [7]. The moderation that is needed is helped by defining peer roles, which can help moderators set expectations for participants in an interaction [6].

A Politeness Framework

Beyond ensuring student safety, how can appropriate communication be defined? Can politeness be codified, and even if it can, is it important in online interactions? Researchers Lewis and Paola drew from Brown and Levinson's classic linguistic work *Politeness: Some Universals in Language Usage* when designing the language a chat-based intelligent tutoring system would use [10]. In Brown and Levinson's framework, people across all cultures have a positive and negative "face." The positive face desires to be attractive to others; the negative face desires autonomy. In this framework, politeness is redressing these desires in one's audience and avoiding actions that threaten them. Tutoring inherently involves corrective and instructive statements, which naturally threaten the negative and positive faces, respectively; however, Lewis and Paola were able to draw from strategies suggested by Brown and Levinson to minimize the sharpness of face threats in ther intelligent chat tutor. They assert that politeness is essential for maintaining student motivation.

Lessons from Other Fields

The practices of effective in-person tutors also offer some insight into what makes an effective online peer-support session. Some relevant traits of good tutors are that they listen carefully, offer encouragement, and provide ample time after asking a question (called "wait time") for the student to think [11].

The field of technical editing can also lend its key principle: it is critical to consider the experience level of the student [1]. Choosing to use vocabulary that is known only to those with experience with a subject can be detrimental to learning.

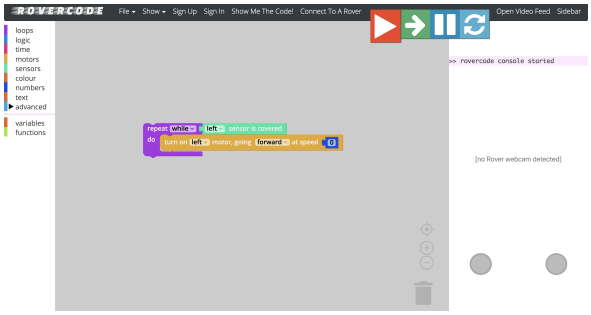


Figure 2: Rovercode's Mission Control workspace.

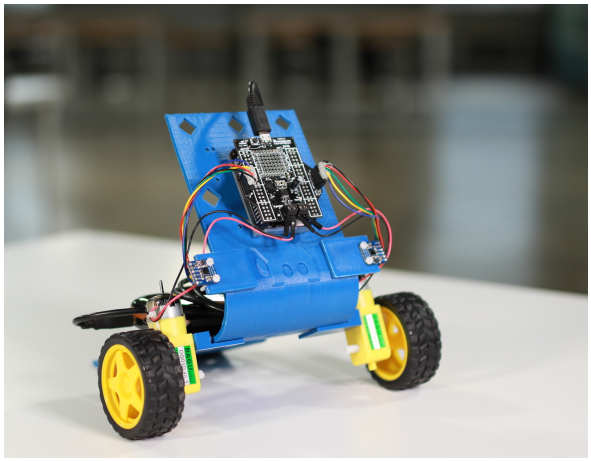


Figure 3: A Rover.

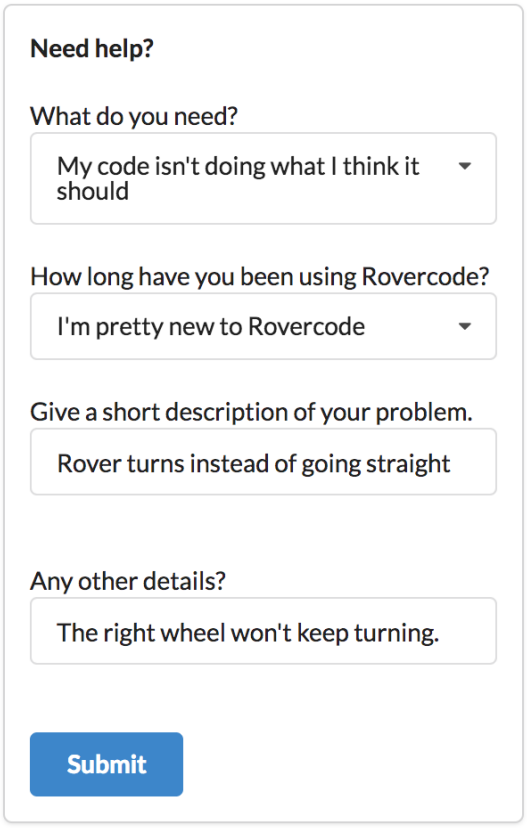


Figure 4: The support request form.

DESIGN OF A SUPPORT SYSTEM FOR ROVERCODE

The principals established above were used to design a synchronous, in-context peer support platform within the Rovercode online programming environment. This section gives an overview of Rovercode, then describes and justifies the features of the support platform.

What is Rovercode?

Rovercode is an open-source educational robotics programming platform. It consists of two parts: a small, wheeled robot (called a *Rover*, shown in Figure 3), and a cloud-based coding environment (accessible at *https://rovercode.com*). Students create block-based programs using the online coding environment called Mission Control, shown in Figure 2. The programs execute in the browser, sending motor commands to the rover and reading sensor data from the rover.

Rovercode was chosen to house the peer-support platform for two reasons. First, it is developed by an open-source community, which makes it easily extendible. Secondly, its cloud-centered architecture facilitates interactions between users well.

Support System Design

The Rovercode support system designed here allows a user who is facing difficulty to create a support request using the form shown in Figure 4. Once the support request has been created, it becomes visible to the community of peers, who can choose to volunteer their help. When a peer from the community initiates an action to provide support on the indicated support request, the two students are joined in a chat session, shown from each student's perspective in Figure 5. Either student may end the support session at any time, and the user requesting support may choose to close the issue or leave it open for another supporter to join.

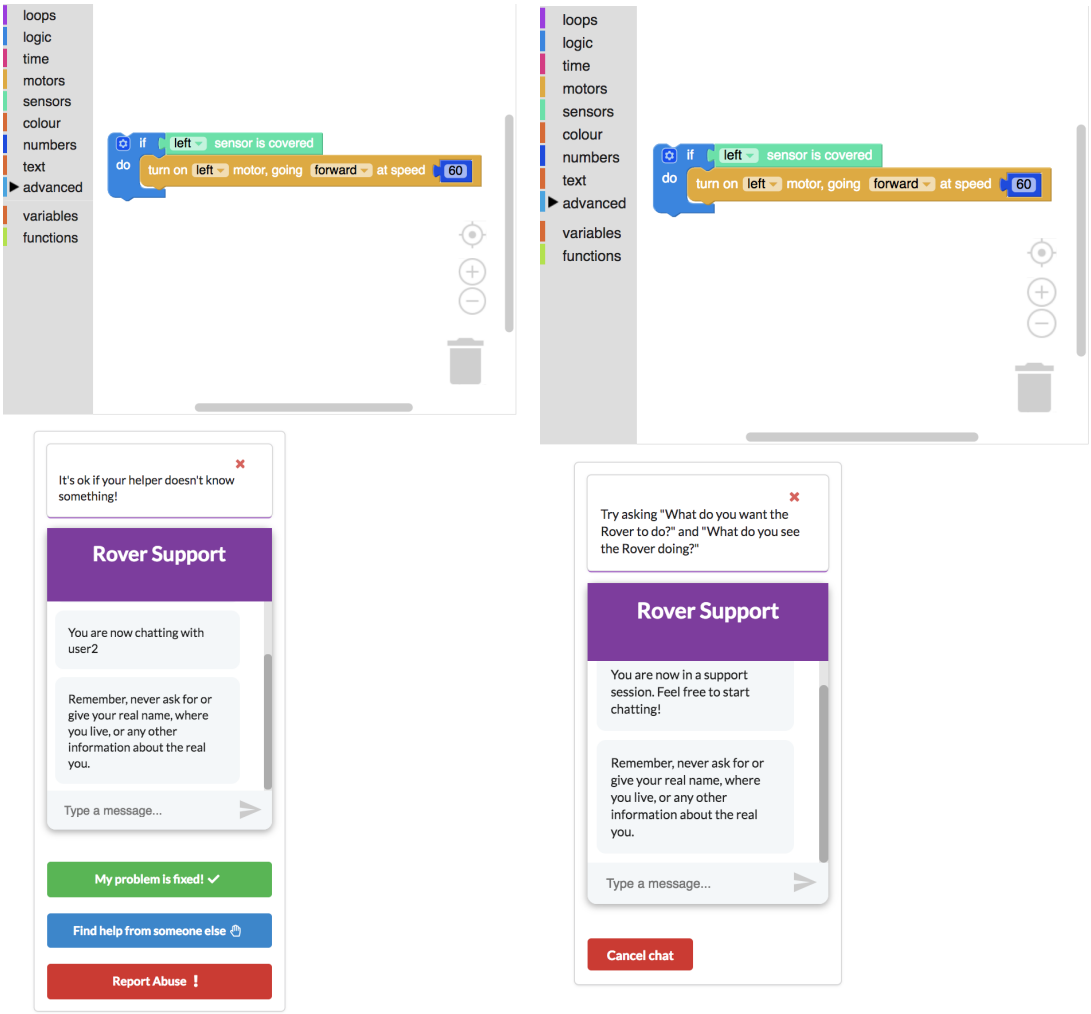
The sections below describe and justify each of these steps in more detail.

Connecting with a Support Provider

A student can request support using a form in Figure 4 within the Mission Control interface. Like a question form on a support forum, this form asks for a topic and an explanation. However, the text entry boxes and intentionally small, suggesting that the user need not capture every detail of the situation.

Once the support request is created, it becomes visible on the homepage for other users. Another user can read the request and volunteer to provide support.

The research above influenced the design of the form in several ways, which are described below.



(a) The view of the user requesting support.

(b) The view of the user providing support.

Figure 5: The views of the users asking and providing support.

Making Purpose Explicit

Fields and Pantic suggested that Scratch has a relatively low rate of abusive communication because interactions on that platform have an established purpose [7]. To attach explicit purpose to support requests, the form in Mission Control includes a drop-down menu to specify the purpose of the request, selected from:

- My code isn't doing what I think it should.
- I don't know what to do next.
- I'm looking for fun ideas.

The selected purpose is visible to potential support providers and helps set their expectation should they choose to volunteer.

Communicating User Experience Level

In the form, users can also select their experience level with Rovercode, one of:

- This is my first time on Rovercode
- I'm pretty new to Rovercode
- I've used Rovercode for a while
- I'm a Rovercode expert

The experience level is visible to potential support providers and allows them to tailor their language as suggested by Amare [1].

Futhermore, prompting a student to recognize her increasing expertise may encourage her to view her role as a mentor and teacher as well as a user, even if that role is not officially assigned. This should create some of the positive effects of roles observed by Fernando in Scratch [6].

Synchronous Communication in Mission Control

Live View of Code: Interacting with Work Products

When two students are connected, they are not connected in some separate support environment where only the question details are visible. Instead, the two users are connected inside the requesting user's Mission Control view – within the context of the problem. While the users are chatting, the requesting user's code blocks are visible to both users, and changes made by the requesting user are seen in real time by the support provider.

This in-context interaction satisfies the principle of social learning theory that students should interact not only with each others' knowledge, but with the products of each others' work. The rich context immerses both students in details and nuances that would be absent from an isolated forum post. The support provider may learn something from a coding pattern he observes in a section of the program that is unrelated to the question at hand.

When connected, only the user requesting support can edit the code; the supporter has a read-only view. The decision not to make the code editable by both users is a compromise: it slightly limits the ability of the supporter to freely interact with the work product, but it encourages the owner of the code to understand suggestions well enough to make the changes herself.

Live Chat: Synchronous Meaning Making

Once connected by the chat interface, students are free to begin discussing the code and the problem: what the user wants to happen, what the user is seeing, and what the user was thinking when she designed the code. Through this interface, users can quickly clarify their ideas and follow directions that were not imagined when the support request was originally posted. It is this rapid, iterative group meaning-making that is the principal benefit of synchronous communication for support. It eases the burden of capturing all relevant information in the support request, which is a primary challenge on forums.

Tip Cards for Encouraging Effective Support

Language and Strategy Coaching

To encourage users toward the best practices for tutoring and politeness strategies discussed above, the strategies were distilled into tips that are displayed on cards through the the interface.

1. **Try to avoid the word "just", as in "you just need to...". Something might be easy for you but a challenge for a first-timer!** This first tip is designed to ensure that providers use language that considers their audience's experience [1]:
2. **Try asking "What do you want the Rover to do?" and "What do you see the Rover doing?".** These questions prompt the requestor to evaluate her own experience and the provider to reinterpret that experience, which are important steps in group meaning-making in informal learning [21]
3. **Do your best to read what the other person wrote carefully. There's no need to guess what they are thinking!** Careful listening is a key trait in effective tutors [11].
4. **Try to be encouraging! Believe that they can do it!** Encouragement is a common trait in effective tutors [11] and redresses the requestor's positive face [10].
5. **Try to be patient after asking a question! If you can, give them some time to think.** Alison asserts that sufficient wait time after a question is important for encouraging critical thinking [11].
6. **Think about using the phrases "Now you might want to..." and "What I would do is...".** Avoiding direct commands redresses the requestor's negative face [10].
7. **Feel free to ask lots of questions!** Socratic questioning is another strategy for redressing the negative face [10].
8. **You're both in this together! Can you use the word "we" sometimes?** Establishing a joint goal is a strategy for politeness [10].
9. **Sometimes, try to help them think from the Rover's perspective. Like, "the Rover didn't know you wanted it to...".** This tip encourages the conventional indirectness strategy for politeness [10].

The tips follow the practice of other online social networks by using "they" as the singular third-person pronoun. It is also worth noting that the guidelines contained in these tips were applied to the phrasing of the tips themselves. For example, the sixth tip reads as a suggestion and not a command.

Moderation and Safety

The design decisions already described serve to tightly couple communications with context and purpose, which has shown to be effective in reducing rates of abusive messages. However, it is still important to provide students with means to block new messages from users who behave inappropriately. The Rovercode support interface provides the "Report Abuse" button, which ends the chat session, logs the chat transcript for moderators, and blocks those users from connecting on any future support sessions. It also directs the users to the Rovercode code of conduct, which describes how moderators will evaluate and respond to the report.

In addition to the reporting feature, there is a message displayed at the beginning of each support session reminding both parties never to share their real names or any other personally identifying information.

EVALUATION

The interface was evaluated by a a middle-school science teacher, a Rovercode contributor, a high-school senior and high-school freshman. In each trial, the evaluator took the role of requesting and providing support in turn. Most of the evaluators noted the tip cards, and several used the recommendations to help guide their tutoring. All of the evaluators requested practical changes, like simplifying the notification that the other user is editing the code or moving the chat box to a more convenient location. One of the high-school students asked for the ability to place markers in the code workspace to signal to the other user.

When asked if they imagined themselves using the support request feature regularly, all of the evaluators said yes. When asked if the imagined themselves providing support regularly, all of the evaluators said that they would, but most on the condition that a reward system is added.

FUTURE WORK

Incentives to Provide Support

As described earlier, social gratification provides non-negligible motivation to provide support on synchronous support platforms. Future research will investigate what extrinsic motivators could provide even more motivation. Common practices include badges and point systems that unlock features, but these must be evaluated in practice to uncover any negative side effects on the community.

The Rovercode platform is usable in classrooms even without built-in extrinsic motivators, because teachers can motivate using the existing reward structure in their classrooms.

A Moderator View

This design does not discuss a view for moderators to evaluate and act on abuse reports. Further research will be needed to create an interface that allows moderators to efficiently and accurately process these reports.

Continued Evaluation

The support platform is currently in the alpha stage in the Rovercode project. When it is deployed in the mainline Rovercode application, continued evaluation will be needed to judge the effectiveness and safeness of the student communications on the platform.

CONCLUSION

This paper showed how constructionist and social learning principles can be used alongside practical lessons from existing online communities to create a synchronous, in-context peer-support platform. The design of such a platform was demonstrated the Rovercode project. In that design, many principles were leveraged in a way that would be impossible on an asynchronous, encyclopedic forum-based support platforms.

REFERENCES

1. Nicole Amare, Barry Nowlin, and Jean Hollis Weber. 2011. *Technical editing in the 21st century*. Prentice Hall Boston, MA.
2. Cecilia R. Aragon, Sarah S. Poon, Andrés Monroy-Hernández, and Diana Aragon. 2009. A Tale of Two Online Communities: Fostering Collaboration and Creativity in Scientists and Children. In *Proceedings of the Seventh ACM Conference on Creativity and Cognition (C&C '09)*. ACM, New York, NY, USA, 9–18. DOI: http://dx.doi.org/10.1145/1640233.1640239
3. David Boud, Ruth Cohen, and Jane Sampson. 1999. Peer Learning and Assessment. *Assessment & Evaluation in Higher Education* 24, 4 (1999), 413–426. DOI: http://dx.doi.org/10.1080/0260293990240405
4. Janet Carter. 1999. Collaboration or Plagiarism: What Happens when Students Work Together. *SIGCSE Bull.* 31, 3 (June 1999), 52–55. DOI: http://dx.doi.org/10.1145/384267.305848
5. Justin Cheng, Cristian Danescu-Niculescu-Mizil, and Jure Leskovec. 2014. How community feedback shapes user behavior. *arXiv preprint arXiv:1405.1429* (2014).
6. Champika Fernando. 2014. Online Learning Webs: Designing Support Structures for Online Communities. (2014).
7. Deborah A. Fields, Katarina Pantic, and Yasmin B. Kafai. 2015. "I Have a Tutorial for This": The Language of Online Peer Support in the Scratch Programming Community. In *Proceedings of the 14th International Conference on Interaction Design and Children (IDC '15)*. ACM, New York, NY, USA, 229–238. DOI: http://dx.doi.org/10.1145/2771839.2771863
8. Glitch. 2018. (2018). https://glitch.com/
9. W Hoppitt and K. N. Laland. 2013. Social learning: An introduction to mechanisms, methods, and models. *Princeton University Press* (2013). http://press.princeton.edu/chapters/s10047.pdf
10. W Lewis Johnson and Paola Rizzo. 2004. Politeness in tutoring dialogs: "Run the factory, that's what I'd do". In *International Conference on Intelligent Tutoring Systems*. Springer, 67–76.
11. Alison King. 1997. ASK to THINK-TEL WHY: A model of transactive peer tutoring for scaffolding higher level complex learning. *Educational psychologist* 32, 4 (1997), 221–235.
12. Cliff Lampe and Erik Johnston. 2005. Follow the (Slash) Dot: Effects of Feedback on New Members in an Online Community. In *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work (GROUP '05)*. ACM, New York, NY, USA, 11–20. DOI: http://dx.doi.org/10.1145/1099203.1099206
13. Lora Oehlberg, Wesley Willett, and Wendy E. Mackay. 2015. Patterns of Physical Design Remixing in Online Maker Communities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 639–648. DOI:http://dx.doi.org/10.1145/2702123.2702175
14. Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.
15. Stephanie M. Reich, Rebecca W. Black, and Ksenia Korobkova. CONNECTIONS AND COMMUNITIES IN VIRTUAL WORLDS DESIGNED FOR CHILDREN. *Journal of Community Psychology* 42, 3 (????), 255–267. DOI:http://dx.doi.org/10.1002/jcop.21608
16. Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay

Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (Nov. 2009), 60–67. DOI:http://dx.doi.org/10.1145/1592761.1592779

17. Amnon Ribak, Michal Jacovi, and Vladimir Soroka. 2002. "Ask Before You Search": Peer Support and Community Building with Reachout. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work (CSCW '02)*. ACM, New York, NY, USA, 126–135. DOI: http://dx.doi.org/10.1145/587078.587097

18. Petr Slovak, Katie Salen, Stephanie Ta, and Geraldine Fitzpatrick. 2018. Mediating Conflicts in Minecraft: Empowering Learning in Online Multiplayer Games. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 595, 13 pages. DOI: http://dx.doi.org/10.1145/3173574.3174169

19. M. Smith and Z. L. Berge. 2009. Social learning in Second Life. *Journal of Online Learning and Teaching* 5, 2 (2009), 439.

20. Joel Spolsky. 2011. The Wikipedia of Long-Tail Programming Questions. (5 1 2011). https://goo.gl/ZSuYEd

21. Mary F. Ziegler, Trena Paulus, and Marianne Woodside. 2014. Understanding Informal Group Learning in Online Communities Through Discourse Analysis. *Adult Education Quarterly* 64, 1 (2014), 60–78. DOI: http://dx.doi.org/10.1177/0741713613509682