Exploring Design Space through Remixing

YUE HAN AND JEFFREY V. NICKERSON, Stevens Institute of Technology

1. INTRODUCTION

Open innovation [Chesbrough 2006] sometimes performs better than traditional company-based innovation [Hippel and Krogh 2003, Poetz and Schreier 2012]. Open innovation communities promote collective intelligence by providing members platforms to design and create use-generated content. Among these communities, many have incorporated a system to encourage users to modify each other's work in a process called remixing [Lessig 2014]. Remixing continues to evolve in open innovation communities including Github [Dabbish et al. 2012], Scratch [Hill and Monroy-Hernández 2013], Thingiverse [Kyriakou and Nickerson 2014], and ccMixer [Cheliotis and Yew 2009]. Remixing in online communities relies on a small number of technical features [Nickerson 2014], and understanding the way these features are used may help us design more productive online communities. Remixing can be thought of as a form of product evolution [Nickerson and Yu 2012]: Together a community explores a design space of possibilities, starting from the many designs already created.

2. EXPLORATORY STUDIES

In our first study, we analyzed projects on Scratch.mit.edu to understand how users explored the design space. Scratch is a programming language created by the Lifelong Kindergarten Group at the MIT Media Lab [Resnick et al. 2009]. Scratch.mit.edu is an online community in which users, mainly youth, create, share and remix projects using Scratch. The projects selected in our study are created after 2013, when Scratch 2.0 was launched, because the raw project code is readily available in text format, allowing us to compute distance measures. For robustness, we also conducted similar explorations on GitHub.

On Scratch, projects are automatically marked as remixes if they are modified based on another project [Monroy-Hernández et al. 2011]. An original project and all of its remixes form a remix tree. Since the project source code can be extracted as text, we use string edit distance [Ristad et al. 1998] as the measure to calculate the distance between any two projects within a remix tree. The distance matrix determines how far a remix has moved in the design space. The multidimensional scaling plot of each entire remix tree based on the distance matrix creates a clear map of the occupation in the design space. Figure 1 shows four typical design space exploration maps of the remix trees.

The exploration maps reveal several interesting phenomenon. In a previous study we noted reciprocal behavior in remixing: sometimes users alternate in modifying each other's work [Han and Nickerson 2013]. In this study, while comparing different remix trees, we found that some trees have a larger portion of distant remixes while other trees contain many remixes that are very similar to the original project. Looking at specific remixes, we can see that some users move much further from the design they remix (nodes 2,4,6), while others stay close to the designs they remix (nodes 1,3); some users will move far from a design but shift their direction back towards the original project (nodes 5,7). Understanding the formation of these different patterns can help us better utilize the remixing process to guide users in making further explorations in the design space.

The different patterns among remix trees are related to the nature of the project. Scratch projects can be roughly categorized into several areas, including games, coloring contests, and animations. Games are often more complex, and are usually structured into levels, which allows for a natural modularity. Coloring contests do not have levels. In some cases they present different characters to be colored, and this can sometimes lead to a different form of modularity.

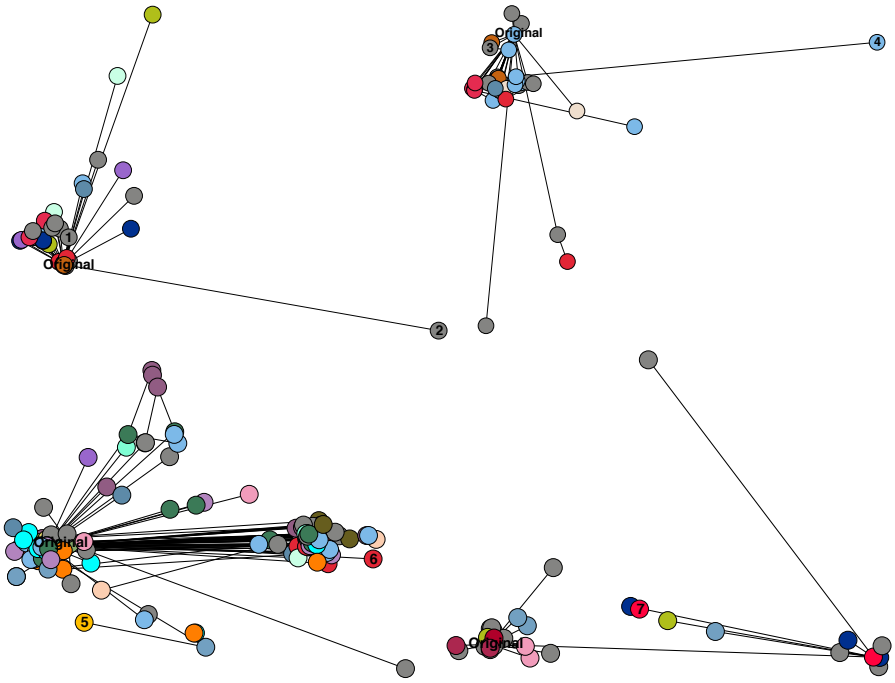

Fig. 1. Design space exploration maps of four remix trees. The MDS plot is created based on the distance calculated by comparing the code difference between every pair of projects within a remix tree. Each node represents a project within its remix tree. The projects are filled with the same color in a remix tree if their creators are the same. The location of the original project is marked by text in each map.

Coloring contest changes tend to be superficial, relating to the colors of shapes, whereas changes to games may involve adding entire levels. Animations have qualities of both, in that the animations sometimes have episodic story structures as well as many characters to be drawn. Thus, genre structures may influence remixing patterns, and perhaps remixing patterns lead to alterations in genres: coloring contests with multiple characters allow for more parallel forms of collaboration.

How do individual participants learn from remixing? In a second study, we analyzed how users explore on their own after remixing other projects. We compared the remix and the projects created after this remix. By extracting the common substrings, we found that some users expand their repertoires by reusing code after remixing other user's projects. Figure 2 is an abstracted model based on examples we found in both Scratch and GitHub.

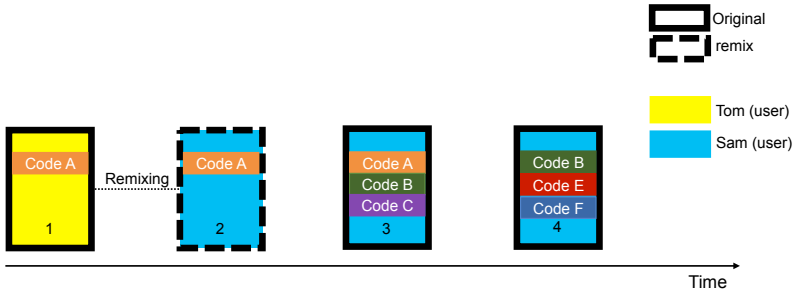

Fig. 2. Individual design space exploration in post-remixing period: users reuse code after remixing.

In this model, Tom first created an original project 1. Then Sam created a remix (project 2) based on project 1. Project 2 shares some common pieces of codes with project 1 as it is a direct remix. Later,

Sam created his own original project – project 3, reusing the same piece of code (code A). After some time Sam created a new project (project 4). This time, he reuses some of his own original code (code B) from project 3. When code A passes from project 1 to project 2, it may widen Sam's set of tools, and encourage him to create his own project. In other words, it might serve as a scaffold. The scaffolding may only need to be temporary, as project 4 consists of all original code. These studies suggest the importance of modularity in remixing, and the sensitivity of modularity to genre.

What happens on projects with predefined explicit modules? In a third study, we looked at projects in MIT's Climate CoLab [Introne et al. 2011], in which users collaborated to create policies related to climate change. The proposals are built around a template that defines explicit modules.

One example of a collaboration is shown in Figure 3. It demonstrates a process of building out the content of modules through additions and deletions (the large changes), followed by a slow process of refinement and convergence, all quantified through string edit distance.

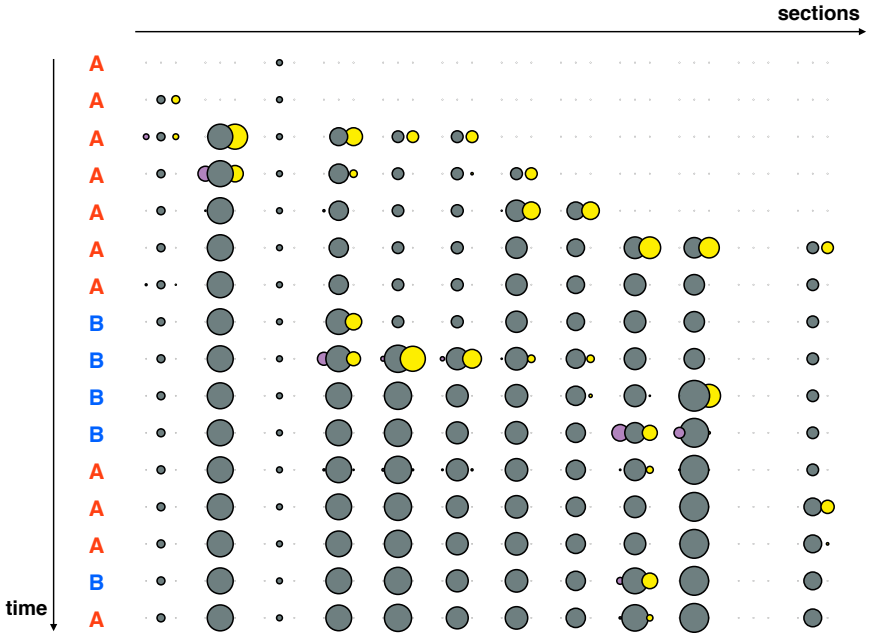

Fig. 3. Users A and B, red and blue, collaborate on 12 different sections of a proposal. Yellow indicates additions of code from the previous version, and purple indicates deletions.

3. CONCLUDING THOUGHTS

Our exploratory studies suggest that remixing is a way of exploring a design space. For the group, it allows for a widespread exploration of a design space, with some individuals searching near and some far. For the individual, remixing may be a way of learning through imitation, eventually leading to new skills and more original contributions. For those designing remixing platforms, explicit modularity in definitions of projects may be a way to encourage parallel work. Given a better understanding of the way creative work happens in remix communities, it may be possible to accelerate this work through recommendation systems that take into account the types of projects, the history of individuals, and the current boundaries of the design space [Ozturk and Han 2014]. This may accelerate both individual skill development and collective exploration.

Acknowledgements
This material is based upon work supported by the National Science Foundation under grants IIS-0968561, IIS-1211084, IIS-1422066, and CCF-1442840.

REFERENCES

Cheliotis, G., & Yew, J. (2009). An analysis of the social structure of remix culture. In Proceedings of the fourth international conference on Communities and technologies (pp. 165-174). ACM.

Chesbrough, H. (2006). Open innovation: a new paradigm for understanding industrial innovation. Open innovation: Researching a new paradigm, 1-12.

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: transparency and collaboration in an open software repository. In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (pp. 1277-1286). ACM.

Han, Y., & Nickerson, J. V. (2013). Remix Networks in Scratch. In Workshop on Information in Networks.

Hill, B. M., & Monroy-Hernández, A. (2013). The remixing dilemma: The trade-off between generativity and originality. American Behavioral Scientist 57(5), 643–663.

Hippel, E. V., & Krogh, G. V. (2003). Open source software and the "private-collective" innovation model: Issues for organization science. Organization science, 14(2), 209-223.

Kyriakou, H., & Nickerson, J. V. (2014). Collective Innovation in Open Source Hardware. Collective Intelligence.

Introne, J., Laubacher, R., Olson, G., & Malone, T. W. (2011). The Climate CoLab: Large scale model-based collaborative planning. In Collaboration Technologies and Systems (CTS), 2011 International Conference on (pp. 40-47). IEEE.

Lessig, L. (2014). Remix making art and commerce thrive in the hybrid economy. Epic Publishing.

Monroy-Hernández, A., Hill, B. M., & Gonzalez-Rivero, J. (2011). Computers can't give credit: How automatic attribution falls short in an online remixing community. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 3421-3430). ACM.

Nickerson, J. V., & Yu, L. (2012). Going Meta: Design Space and Evaluation Space in Software Design.

Nickerson, J. V. (2014). Collective Design: Remixing and Visibility. In Design Computing and Cognition.

Ozturk, P., & Han, Y. (2014). Similar, Yet Diverse: A Recommender System. Collective Intelligence.

Poetz, M. K., & Schreier, M. (2012). The value of crowdsourcing: can users really compete with professionals in generating new product ideas? Journal of Product Innovation Management, 29(2), 245-256.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: programming for all. Communications of the ACM, 52(11), 60-67.

Ristad, E. S., & Yianilos, P. N. (1998). Learning string-edit distance. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 20(5), 522-532.