# Iterative R&D in Education

## Introduction

This document describes the iterative research and development process Enlearn used during the EF+Math program to design and test new educational software tools, classroom protocols, and paper-based learning activities. Rather than focusing heavily on a small number of fully polished ideas from the outset, Enlearn instead developed dozens of lightweight prototypes and put many of them in front of learners early, using small-scale tests to understand how students responded and what they were actually learning. Using this method, the project developed around ~50 early stage prototypes that were piloted with a small number of users, and ~15 fairly-developed prototypes that were used with significant numbers of users.

By cycling rapidly through brainstorming, prototyping, and user testing, Enlearn aimed to learn as much as possible, as early as possible, and to focus more deeply on the ideas that showed real promise for supporting students' thinking and learning. We hope some or all the elements of this process can be useful to other teams who want to try rapid, wide-breadth iterative R&D.

## Process Overview

Our process looks like the following:

1. **Set a goal**: In this example, we will use the goal: "Help students develop cognitive flexibility while working on math."

2. **Choose a rough hypothesis and mechanism**: In this example, our hypothesis and mechanism is: "Reasoning about how other people are doing math requires cognitive flexibility. Therefore, practicing tutoring or communicating with other people will improve cognitive flexibility."

3. **Brainstorm:** Create 10+ possible prototype ideas following the mechanism. Taking cognitive flexibility as an example, we would think about any activity requiring students to try to solve a problem in different ways - perhaps even incorrect ways! To help us brainstorm, we tried to think about all the things students do, or could do, in everyday situations that had any resemblance to our mechanism (practicing tutoring or communication). Then we tried to find ways for students to practice those skills in a way a computer could support. Some we came up with:
    a. Students often help other students, so perhaps they could solve multiple choice questions about students to debug incorrect student work. This would be one way to get students to intentionally "do" incorrect math, which inherently requires more cognitive flexibility because it increases the number of possible strategies.

   b. Students would need to take the Wisconsin Card Sorting Task in our experiment so that we could measure their cognitive flexibility. So perhaps we could design a mathematical version of that task using fractions instead of colors and shapes.
   c. Students are often asked by teachers to explain their thinking. Perhaps we could ask students to create mathematical explanations following a rubric describing qualities of a precise and useful mathematical explanation. This would clearly require some cognitive flexibility because it would require students to think about how the listener might understand a math concept.

4. **Prototype:** Select ~5 of the most promising ideas and develop "cheap" versions testing the basic principle. Almost all ideas can be adapted to paper activities requiring no coding at all, but those requiring computer or other participant interaction can be done by having the facilitator (or "experimenter") simulate what a computer or other human would do, called Wizard of Oz (Ramaswamy 2022).

5. **Initial user test:** Test the prototypes with 3-5 participants (or "users") and note what worked or didn't work. We use such a small number because our first goal is to iron out the obvious problems, and if a problem is obvious then it should show up within the first few people to test it. The important question we ask in the initial tests is what students are learning from the experience, *not* whether the questions were worded well, or the paper interface made sense, or the artwork was good. The experimenter has to be willing to interject to keep the user on track without short circuiting the "core" idea. For example, we may reword a question the user didn't understand or suggest they pretend to click a button to continue, as these problems can be fixed with better design. However, we wouldn't help them solve a problem with a back-and-forth conversation, unless the prototype is testing whether a specific kind of conversation will help users learn.

6. **Iterate**: Repeat the brainstorm, prototype, and initial user test as many times as necessary until we feel confident one of the ideas is likely to work.

7. **Experiment**: Develop a "real" version of the prototype. Generally this will be as low-cost as possible with the goal that users can do it without experimenter intervention. This means user interface, experience, feedback on wrong answers, etc. are important; however, art and polish are not important. Then run a test with users with the prototype and possibly control(s), then analyze the data to learn if the prototype does in fact achieve our initial goal.

## Iterative R&D team "requirements"

The true requirement is that the team has the "spirit" of wanting to figure out what approaches might be promising, in the shortest time possible, and doesn't get overly attached to any

particular idea. However, our R&D process does go smoother if certain requirements are met. We have found that small teams (~5 people), iterate faster than large teams, mostly because they don't get bogged down by committee design. It also helps a lot to have members with the following skills on the core iteration team:

- UX person who knows how to mock up an idea in low-cost ways
- Coder who can write interfaces quickly and with low cost, if a computer is *necessary*
- A person with an education background that has a sense of what approaches might work with students
- A researcher who knows what has already been tried and what interesting research questions are to answer
- An outreach person who can recruit small numbers of users and run the tests

Oftentimes, one person fills multiple roles, and indeed it is often better to have one person with skill overlap than two separate experts. For example, a coder who has some education experience can decide how to code the feedback on wrong answers as they build their test interface that are more likely to be helpful to the first testers; this way, you get a better sense of whether the underlying idea of the prototype is good the first time you test it, rather than merely learning that the user had no idea what the feedback meant and as a result couldn't answer any of the questions.

## Why iterative R&D?

Why do we develop this way instead of more carefully designing one or two activities and polishing them really well before asking participants to try them? At least 75% of our ideas never make it past the brainstorming phase, and a similar number fail after the initial user tests - sometimes after having undergone several changes to make them easier for users to understand. That seems like a lot of work that never makes it to the light of day.

### The marshmallow test (no, not that one)

To answer this question, let's consider other examples of design challenges. Peter Skillman and Tom Wujec created a design challenge where participants were given one marshmallow, one meter of string, one meter of masking tape, and twenty pieces of uncooked spaghetti. Groups had 20 minutes to create a stable structure supporting the marshmallow as high as possible.

Who was the best at this task? Architecture students, because they knew how to build things. Second best was kindergarteners. All the other groups were worse, including CEOs; interestingly, CEOs with executive assistants did better. The worst groups were business school students. Kindergarteners were best at the task because they spent almost all their time experimenting and learning about the structural properties of tape and uncooked spaghetti. Business students were the worst because they spent most of their time deciding who was in charge and planning before building (Skillman, 2019).

### Iterative prototyping in research

This idea of rapid iterative development is not just the domain of TED Talks and business consultants. Research on solving design problems also supports the efficacy of rapidly iterating with “cheap” prototypes first before trying to solve a problem “for real.” In an egg drop experiment where participants have to use common materials to keep an egg safe from a fall (Dow, Kate, and Scott 2009), participants who were asked to spend the beginning of their time just building different designs did better than those who spent the beginning of their time planning the best possible design. And in online web advertisement, participants who tried several ads at the same time and received feedback in parallel created better and more varied ads than those who designed ads more carefully one at a time receiving feedback after each one (Dow et al. 2010).

## Why iterative development in education?

Clearly, iterative development is not always appropriate. An aerospace company cannot just try 10 full rocket designs with no planning and pick the one that’s best. Nor can a school try 10 different sets of grading policies in the same year and choose the best one. However, educational software or protocol design is a good target for rapid prototyping because:

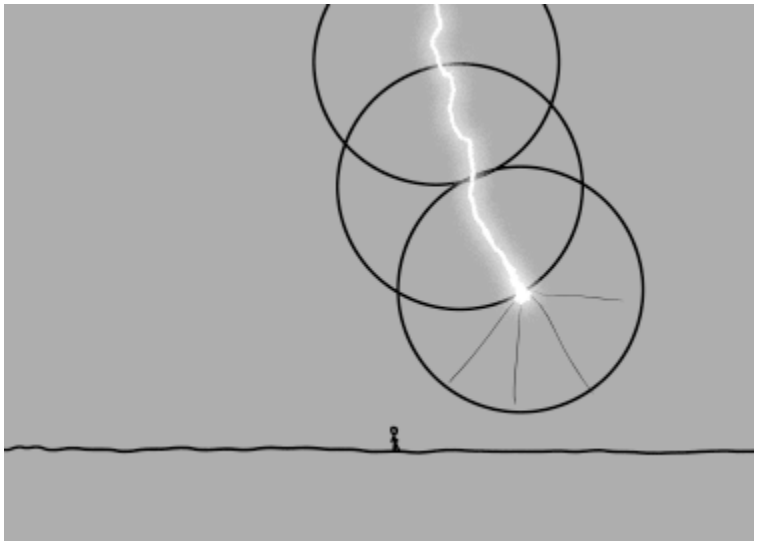
A. The difference between a mediocre idea and a good idea is huge.
B. We can’t predict how learners will react to an idea without trying it.
C. Actually building and deploying a full version of an idea is expensive.
D. It is possible to build the “core” of an idea in a low-cost way, especially if the experimenter can simulate computer decisions.
    a. If you need inspiration, game designers excel at paper prototyping games that seem like they require a computer (Mignano 2016).
E. Testing an idea *with a small number of people* is cheap.
    a. For early prototypes, usually 3-5 people is enough to understand what the major sticking points are and whether the core idea is working.
    b. The time for expensive 30+ user “proper experiments” is later, when you are more confident you have something that will work.
F. Watching people use (or more commonly, fail to use) an idea in person helps the experimenter learn about what types of ideas might work and why.

Using the reasons (A-F) stated above, we can understand why iterative development in education is important: Because mediocre ideas are much worse than good ideas, it’s important we find good ideas during development (A). However, we don’t know ahead of time what ideas work (B). Unfortunately, developing a full deployment version of software is expensive (C), especially compared to low-cost versions given by an experimenter (D). So we cycle through many low-cost ideas with small numbers of users (E), gaining information about what kinds of ideas work or not (F). Eventually we learn enough that we know to abandon the original idea entirely, or are relatively confident that we have an approach that is worth trying in a real experiment.

While different people will think about development in different ways, I personally find it helpful to think about R&D in the “mastery” vs “performance” mindsets (“Goal orientation”, n.d.) popularized by Carol Dweck’s growth mindset work. We want our students to focus on practicing and growing their brains, not final grades or proving they’re smart - and in doing so, their final grades improve more than if they focused on grades in the first place (Blackwell, Trzesniewski, and Dweck 2007, 246-263). If we were to take our own advice, then we should prioritize trying and learning about designing good educational experiences - growing our instructor brains - rather than being overly concerned about building the perfect educational tool right off the bat. Ironically, this leads to us designing better tools in the end.

## Inspiration as a lightning strike

We often describe inspiration as striking like lightning in the sense that it’s sudden and dramatic. This is, of course, nonsense - people get better at skills through practice, not random luck from nowhere. However, there *is* another sense in which lightning strikes are a good metaphor for finding good ideas.



Lightning searching for a path (Munroe, n.d.)

Lightning does not work the way it looks like it does. First, there is a “leader” strike with low current that searches for a path from the cloud to the ground. Roughly speaking, it does this by “looking” in a ~30m radius around itself, jumping randomly to an area with higher positive charge. Since the ground is positively charged, this is usually towards the ground. It then repeats this process until eventually it “sees” the ground and jumps to the ground. At this point, the lightning bolt you see is actually a return stroke that happens once the full path has been established, and is much stronger than the leader.

Educational software has a similar property to lightning in that we generally know where we want to go, but don't have a crystal clear plan to get there. If we simply pick a random direction and spend all our resources in that direction, we have equal odds of ending up further than we started. Thus it makes more sense to look all around us by trying a bunch of different ideas, gathering information about how those ideas work in practice. We can then pick the best ideas and repeat the process, and as we gain information we will start picking ideas that go in the right direction until we find something that is really helping students. At this point, the true "lightning strike" happens where we get to build a really nice, polished piece of software or curriculum or classroom protocol now that we've established the full path from where we started to where we want to go. From the outside it looks like a genius came up with the full idea off the top of his head, but only because they can't see all the discarded ideas along the way.

References

Blackwell, Lisa S., Kali H. Trzesniewski, and Carol S. Dweck. 2007. “Implicit theories of intelligence predict achievement across an adolescent transition: a longitudinal study and an intervention.” *Child development* 78, no. (1) (January): 246-263. 10.1111/j.1467-8624.2007.00995.x.

Dow, Steven P., Alana Glassco, Jonathan Kass, Melissa Schwarz, Daniel Schwartz, and Scott R. Klemmer. 2010. “Parallel prototyping leads to better design results, more divergence, and increased self-efficacy.” *ACM Transactions on Computer-Human Interaction* 17, no. 4 (December): 1-24. https://doi.org/10.1145/1879831.1879836.

Dow, Steven P., Heddleston Kate, and Klemmer R. Scott. 2009. “The efficacy of prototyping under time constraints.” *C&C '09: Proceedings of the seventh ACM conference on Creativity and cognition*, (October), 165-174. https://doi.org/10.1145/1640233.1640260.

“Goal orientation.” n.d. Wikipedia. Accessed September 9, 2023. https://en.wikipedia.org/wiki/Goal_orientation.

Mignano, Marco. 2016. “Use Paper Prototyping to design your games.” Game Developer. https://www.gamedeveloper.com/design/use-paper-prototyping-to-design-your-games

Munroe, Randall. n.d. “Today's topic: Lightning.” What If? (XKCD). Accessed September 9, 2023. https://what-if.xkcd.com/16/.

Ramaswamy, Sara. 2022. “The Wizard of Oz Method in UX.” Nielsen Norman Group. https://www.nngroup.com/articles/wizard-of-oz/.

Skillman, Peter. 2019. “The Design Challenge (also called Spaghetti Tower): | by Peter Skillman.” Medium. https://medium.com/@peterskillman/the-design-challenge-also-called-spaghetti-tower-cda62685e15b.