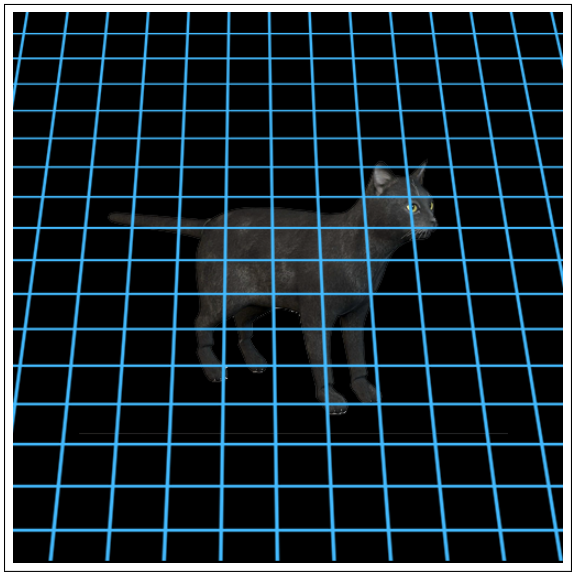
Community Systems as a Learning Tool and Incentive:

How a homegrown community and infrastructure can encourage students to learn outside of class while encouraging camaraderie and the adoption of the open source spirit

Josh Natis



For Kiwi

1 Rose-tinted Glasses

Unfortunate circumstances, and what we can glean from the simplicity of the past.

As a student, it's easy to feel useless in the current state of the world's software ecosystem. At times, it seems like everything has been invented already. For the most part, we're only able to program "toy" projects, and if we do decide to be amicable and share them with the world, our code falls upon deaf ears – there is no positive reinforcement for our feedback loop, our programs do not seem to help anybody. GitHub is brimming with programs, why should anybody be concerned with ours? Projects we care about are so complex that we can hardly grok their code, let alone offer any meaningful help.

Looking far into the past, the picture is less bleak. Programmers were a scarce resource. There was no internet, and thus no gigantic repository of programs to render yours obsolete. If you wrote a program, you were contributing to your community's infrastructure, building it up with more and more utilities over time. Every program you wrote bettered the system, extending the capabilities of whomever you were sharing your system with (and you were certainly sharing your system, forcefully thrust into a community, as computers were far too expensive to allot personal computing). Systems themselves were simpler, built from primitives one could reasonably wrap their head around, so adding an impactful change was possible. This endows programming with a sliver of humanity – you are doing a favor to your community by doing this work. In modern day, this is often replaced by an appeal to capitalism – you are improving your resume by programming this, it will help you get a job. This leaves us hollow.

2 Emulating the Past

In my view, there's no reason that these circumstances can't be reproduced. By simulating a small community where all students reside and interact, sharing programs, improving infrastructure and working on the community itself as they chug along, we can replace the hollow drive of modern society with a wholesome ethic of helping one another learn and be better. We all crave this human connection that we are so deprived of – many students find this in games (specifically in multiplayer role-playing games, where you take the form of a fictional character residing online, spending countless hours improving your character and in the process forming meaningful connections with others in the same position). This simulated world allows for emotional release, but all of your hard work only manifests in the game itself. If there was a way to capture the spirit of such games (which are really just distributed, simulated communities) but within computer systems, this could prove an effective way to overcome some of the issues students experience in the modern software ecosystem. This is really open ended, and equally a technological and organizational problem.

3 Attempt at Less Hand-waving

I'm basically thinking of a way to get students to want to program outside of class for a good cause, and make them feel good for doing so (all while learning new stuff). Coming together and improving the infrastructure of the school is a great way of doing so.

For example, some Hunter students including myself created a website to mirror all of the syllabi for computer science courses. Previously, there was no such repository of syllabi, and now that there is the general quality of life of Hunter computer science students has (marginally) improved. At the same time, some of us learned new things and created opportunities for students to make meaningful contributions (by sharing their syllabi or making pull requests on GitHub). In return, we marked students as contributors on our GitHub repository, thus providing positive reinforcement for their choice to help us out. This is a super simple project, but it made a meaningful difference.

There are many things missing from our school, I'm sure students could think of other things. Sometimes small groups of students may come up with solutions to these problems, but don't necessarily share them with the rest of the community because there is nowhere to share. Or, at other times, the school's existing infrastructure is behind closed doors, thus killing off possible innovative solutions (imagine if CUNYFirst had an API that was open for student use? there would be so many cool tools made). Now image if every time a Hunter student made a cool program they tossed it in */usr/local/bin* or uploaded it to a hypothetical CUNY Hunter organization on GitHub. This would have a magnitude of positive effects – just to name a few: (1) many more useful programs!, (2) more source code that other students can read (and maybe be more likely to understand because they share a background with the author), (3) if a student finds a program cool then may message the author and ask them to collaborate or explain, or just generally to discuss, (4) more incentive for students to write programs outside of class, and etc.

We also do not have students working on upkeeping ENIAC as of now. I've found that some universities accept motivated students and teach them how to maintain the school's facilities (like web servers or the available supercomputers). This is a great opportunity for mentor-mentee relationships (either between faculty and students or experienced students and their more junior counterparts).

4 Examples and Shared Sentiments

I've gathered a few examples of infrastructures, both present and past, which I think support/demonstrate this idea.

4.1 Hackers: Heroes of the Computer Revolution

A brilliant book by Steven Levy. The picture he paints of the MIT Hackers encapsulates an ideal milieu. (The content is located in chapters 1 through 7).

4.2 A Conversation From the TUHS Mailing List

> I do remember clearly celebrating the death of ptrace by removing ptrace(2) from the copy of the V8 manual in the UNIX Room. It took up two pages, and they happened to be facing pages, so I glued them together.
I wish it was as easy for others to have such satisfaction these days.
Norman Wilson

>Amen to that. I think we all lived, you Bell Labs people especially, in a simpler time. We were trying to fit into 64K, split I/D 128K, my Z80 was 64K but some extra for graphics, then the VAX came and we were trying for 1MB, Suns with 4MB.
So small mattered a lot and that meant the Unix philosophy of do one thing and do it well worked quite nicely.
What that also meant, to people coming on a little bit after, was that it was relatively easy to modify stuff, the stuff was not that complex.
Even I had an easy time, my prime was back at Sun when SunOS was a uniprocessor OS. That is dramatically simpler than a fully threaded SMP OS that has support for TCP offloading, NUMA, etc, etc.
I really don't know how systems people do it these days, it is a much more complex world.
So I'm with Norm, it was fun back in the day to be able to come in and have a big impact. I too wish that it was as easy for young people to come in and have that impact. I've done that and it was awesome, these days, I have no idea how I'd make a difference.
Larry McVoy

4.3 Richard Stallman on the Development of Emacs

>The interesting idea about Emacs was that it had a programming language, and the user's editing commands would be written in that interpreted programming language, so that you could load new commands into your editor while you were editing. You could edit the programs you were using and then go on editing with them. So, we had a system that was useful for things other than programming, and yet you could program it while you were using it. I don't know if it was the first one of those, but it certainly was the first editor like that.

This spirit of building up gigantic, complicated programs to use in your own editing, and then exchanging them with other people, fueled the spirit of free-wheeling cooperation that we had at the AI Lab then. The idea was that you could give a copy of any program you

had to someone who wanted a copy of it. We shared programs to whomever wanted to use them, they were human knowledge. So even though there was no organized political thought relating the way we shared software to the design of Emacs, I'm convinced that there was a connection between them, an unconscious connection perhaps. I think that it's the nature of the way we lived at the AI Lab that led to Emacs and made it what it was.

It was Bernie Greenberg, who discovered that it was (2). He wrote a version of Emacs in Multics MacLisp, and he wrote his commands in MacLisp in a straightforward fashion. The editor itself was written entirely in Lisp. Multics Emacs proved to be a great success — programming new editing commands was so convenient that even the secretaries in his office started learning how to use it. They used a manual someone had written which showed how to extend Emacs, but didn't say it was programming. So the secretaries, who believed they couldn't do programming, weren't scared off. They read the manual, discovered they could do useful things and they learned to program.

So Bernie saw that an application — a program that does something useful for you — which has Lisp inside it and which you could extend by rewriting the Lisp programs, is actually a very good way for people to learn programming. It gives them a chance to write small programs that are useful for them, which in most arenas you can't possibly do. They can get encouragement for their own practical use — at the stage where it's the hardest — where they don't believe they can program, until they get to the point where they are programmers.
Source: https://www.gnu.org/gnu/rms-lisp.en.html

4.4 NYC Mesh

NYC Mesh is a group of volunteers building a community network and "an infrastructure commons that is accessible to all New Yorkers". You can sign up as a volunteer without much prior knowledge on the subject matter and the organizers will teach you what you need to know during the process. I think this is a good living example of a small community that works for a good cause and educates while they're at it.

Source: https://www.nycmesh.net/

4.5 APIs from MIT

https://ist.mit.edu/apis?category=15
This is an example of making information programmatically open for students to use. If we wanted to make some APIs maybe this would be a good reference point.

5 Is This Possible?

I have no idea. I also don't know if this is a proper topic for an independent study, nor how I could loop operating systems into here. Hopefully there are some good ideas in this document, and maybe we could use them as seeds for more full-formed projects.