EECS 481 HW6b
Michael Jecmen
mjjecmen

PeacefulCraft Network (PCN)

SELECTED PROJECT

I selected the open-source project PeacefulCraft Network (PCN), a Minecraft community with a network of servers and a very small yet regularly contributing dev team. The network has hosted an estimated 32,000 unique players since its inception in 2012, and is beloved by its dedicated core player base. It has a rich set of plugins, exciting community events, and its fun custom game modes -- created by the in-house development team! The dev team consists of 2 veteran staff developers (with PCN since its inception) and a slew of occasional (usually one-off, usually inexperienced) contributors, which made this project stand out like a sore thumb in my shortlist. Though the project is clearly small, these two core developers are extremely capable, and their experience and creativity shows in PCN's many open source repositories. The team has ambition to spare, and the PCN issue tracker boasts >50 open tickets, even with some labeled "good first issue" and "help wanted".

SOCIAL GOOD INDICATION

I do not believe PCN contributes to Social Good, at least in the manner in which Social Good has been outlined by this assignment. To explain this nuance, on the PCN About Page, the owner posits that "PeacefulCraft, the staff team, and our wider user base seek to establish an inclusive community where as many individuals as possible can feel at home." While I personally believe this claim to be genuine, and know that PCN has brought great happiness to the lives of many, at the end of the day PCN is a community founded around Minecraft: enjoyable for its users but does not quite contribute to some greater notion of societal good.

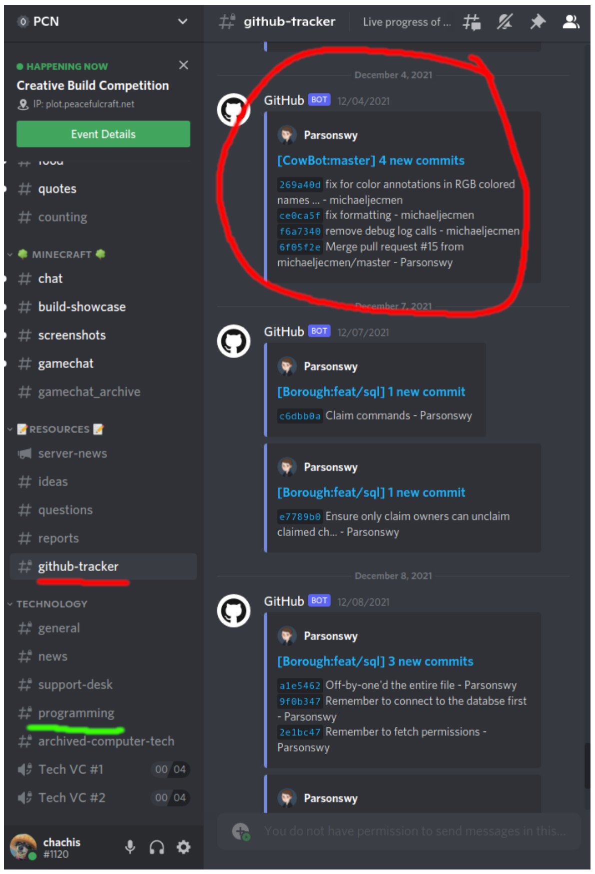
PROJECT CONTEXT

Here's the short history: this project is, and was for the first 6 years of its life, a Minecraft server run by two friends from high school, hosted on a small organization's server that one of the friends worked for (who they'd let use the rack space for free). Then, as things got larger, the community needed to expand, the two owners finally gave in and bought web hosting space for their now multiple-servers. After a certain period of peaking popularity, the two had become quite competent sysadmins and modders, and their now smaller yet seriously devoted player base had a plethora of in-house developed mods, gamemodes and plugins to choose from. This is all to say that business-wise, PCN is a passion project, and it is certainly operated at a loss. The owners don't see it that way, however, which I'm sure is why after all these years they've continued to keep the community free and open!

They aren't actively trying to gain new members or make money, but nevertheless their community has boasted an impressive player count throughout its lifespan. As a result they've regularly asked their community for open source contributions and set up their many GitHub repos to encourage such growth. I've been friends with one of the owners for a long time (we went to high school together), and I wanted to help build something he's passionate about. I believe I'm the first non-owner with significant development experience who has contributed, and I know my contributions will set a good road map for anyone looking to get involved with PCN.

PROJECT GOVERNANCE

The main means of communication between developers and contributors is Discord. PCN has a very active official Discord server (~300 members), which I've included a crudely labeled screenshot of below:



December 4, 2021

GitHub BOT 12/04/2021

Parsonswy
[CowBot:master] 4 new commits
`269a40d` fix for color annotations in RGB colored names ... - michaeljecmen
`ce0ca5f` fix formatting - michaeljecmen
`f6a7340` remove debug log calls - michaeljecmen
`6f05f2e` Merge pull request #15 from michaeljecmen/master - Parsonswy

December 7, 2021

GitHub BOT 12/07/2021

Parsonswy
[Borough:feat/sql] 1 new commit
`c6dbb0a` Claim commands - Parsonswy

Parsonswy
[Borough:feat/sql] 1 new commit
`e7789b0` Ensure only claim owners can unclaim claimed ch... - Parsonswy

December 8, 2021

GitHub BOT 12/08/2021

Parsonswy
[Borough:feat/sql] 3 new commits
`a1e5462` Off-by-one'd the entire file - Parsonswy
`9f0b347` Remember to connect to the databse first - Parsonswy
`2e1bc47` Remember to fetch permissions - Parsonswy

Parsonswy

Red, middle left and top middle: GitHub commits are automatically posted to a channel which serves as a more accessible commit log. My accepted contributions can be seen here as well, under the GitHub name michaeljecmen. There are ~30 development-interested community members who follow this channel, and an active attempt by the PCN devs to foster this interest. I personally find this very neat.

Green, lower left: The programming-specific channel where details about recent commits are discussed.

EECS 481 HW6b
Michael Jecmen
mjjecmen

Once I was added to this server, I communicated with a PCN developer via Discord direct messages whenever I lacked understanding or needed help setting up a certain part of the project. Any informal line of questioning always went through Discord. As an example, here is an exchange we had over Discord DMs while I was attempting to reproduce the bug outlined by CowBot issue #9:

> *<me>: hmmm I dug around to see how to nickname with color annotations via multichat and didn't see anything*
>
> *unless I'm misremembering*
>
> *<**dev**>: <links to a MultiChat docs page where the color annotation commands are enumerated>*
>
> *<me>: ah, if that's the answer than I am probably reproducing the bug*
>
> *I thought those annotations weren't working because I didn't see the color in game* (it turns out the color didn't appear because I was using the hex code for white, which is the default Minecraft text color already)
>
> *but the &x's do come through to discord even though they don't appear in the mc username*
>
> *<**dev**>: Yeah. That is what the ticket was trying to describe.*

This conversation was the result of a bug report lacking in information, and the usual context unfamiliarity of a new contributor.

The process by which contributions are accepted is quite traditional for an open source project, but lacks formally documented requirements and standards -- also somewhat typical for a small operation. Developers must choose a task from the issues page of their repo of choice, fork the repo, make the fix/implement the feature on their branch, and submit a PR which closes the relevant issue. One of the lead devs will review the PR, leaving their thoughts or reservations in the comments of the PR. Some back-and-forth at this stage should be expected. Here's a relevant example of this style of communication from another contributor's PR.

Beyond the actual code changes outlined in each PR, PCN has no formal (documented) requirements for design or quality assurance. These requirements do exist in the minds of the core developers, however, and here are the implicit requirements I gathered from my conversations with them: do not pull new dependencies into the project (unless they are test scoped or approved by a dev), decouple new code into separate modules wherever possible, abide by standard Maven project organization for new code, obey existing style guidelines (the repo uses space-based tabbing, and camelCase function names, for example), and document and write tests for non-trivial code.

EECS 481 HW6b
Michael Jecmen
mjjecmen

Wherever possible, I tried to go “above and beyond” the quality of previous contributions to ensure that my work would have the greatest chance of being accepted.

TASKS

I worked through the first two of the four tasks I outlined for myself in the initial HW6a proposal.

Task #1: Fix for RGB colored player names have color annotations in them (BUG)

Description

PCN uses a plugin called MultiChat to synchronize in-game chat across its many servers. CowBot listens to this chat, and re-posts the messages to the #gamechat channel in the official PCN Discord server. MultiChat also allows players to change their in-game names, and supports RGB color formatting for these nicknames (in the form of a hex code prefixed by “&x”). For example, setting your nickname to “&xff0000first&x00ff00second&x0000ffthird” would yield ”firstsecondthird”. When CowBot relays messages from users with hex code colored nicknames to Discord, the hex codes themselves are cleaned from the relayed usernames, but the “&x” prefixes are not. To be clear, “&xfirst&xsecond&xthird” would be the sender name forwarded to the Discord #gamechat channel.

Most of my time spent fixing this bug was spent reproducing the issue. While MultiChat supports nicknaming with a \nick command, another PCN plugin supports nicknaming with a \nickname command. The latter plugin, as it turns out, does not support colored names. Once I’d gotten down to testing MultiChat, I found it also supports simpler, non-hex codes which use the “&” prefix for coloring (for example “&c” means red). These codes, confusingly, are *not* relayed to the Discord chat -- so not the source of the bug. Once I’d narrowed the bug to specifically MultiChat nicknaming with “&x” hex code coloring, I found the source of the bug (see this comment in the issue thread for my findings). The fix was then a simple case of removing all instances of “&x” from each username.

Submitted artifacts

Pull request (accepted): https://github.com/peacefulcraft-network/CowBot/pull/15

You can find my manual end-to-end test screenshots right there on the main conversation page, and the code changes (with my personal justification comments for my changes here and here) on the changed files page. This is a very small, simple patch, which I think was a great starting point for this project.

Task #2: Add Better Formatting from Discord to Minecraft (BUG + FEATURE)

Description

CowBot also listens for user posts in the #gamechat channel, and cross-posts those messages to the PCN Minecraft chat. When messages are relayed in this way, the poster's Discord rank (e.g. User, Admin, Moderator, etc) is relayed along with the name and contents of the message. Discord allows you to specify a color for each rank, and CowBot *should* keep the color of the rank when relaying the full [rank]+[username]+[content] message. Due to recent changes in Discord's color formatting, however, the relayed rank color is often incorrect when received.

I did not originally have all of this background -- when I initially hopped on a call with a developer to ask him about for a bit more clarification on this issue, he mentioned that it "shouldn't be too hard" to pass along the correct formatting from the message contents as well (preserving italics, bold, strikethrough, etc), and asked that I consider that a part of this issue.

To fix the color formatting issue, I just had to convert the Discord4j Color object into a native Java Color object. Very straightforward. The text formatting fix, however, was anything but straightforward. My first thought was: "there's got to be a library for parsing markdown formatting". Turns out, there are a bunch -- but they're all massive (look at the number of dependencies I'd be pulling in here), written in a different language (like JavaScript, crushingly not the same as Java), or would parse different kinds of Markdown formatting too (Discord only recognizes these kinds of formatting -- something like <h1>header</h1> should be passed along as plaintext). So I ended up writing my own parsing and formatting conversion module, which took quite a long time.

Submitted artifacts

Pull request (pending): https://github.com/peacefulcraft-network/CowBot/pull/16

Color formatting bugfix:

https://github.com/peacefulcraft-network/CowBot/pull/16/files#diff-57982eaa27d7e83143950fd0bdb89dc2d3604c7cc6e388981edddd9d3455b1e5

Color formatting end-to-end manual tests:

https://github.com/peacefulcraft-network/CowBot/pull/16#issue-1072900702

Text formatting translation module:
https://github.com/peacefulcraft-network/CowBot/pull/16/files#diff-1f76dba845d727c4240a669a015b71dfca8f335763a3a530007307ce7e5100f1

Text formatting translation module tests:
https://github.com/peacefulcraft-network/CowBot/pull/16/files#diff-4ccfec78446b9c8f8fb134d9219ca8f10f9c39657f7b113e43968ea4d1cf0a5d

This is a much larger pull request, which is why I asked the developer if I should split it up into 2 PRs (smaller changes are better, each PR should be for a distinct feature -- thinking back to Delta Debugging and the code review modules).

EECS 481 HW6b
Michael Jecmen
mjjecmen

QA STRATEGY & EVIDENCE

As seen here and here, both tasks had end-to-end manual tests which I designed, ran, and uploaded screenshots of to my PRs. The tickets I worked on involved the interactions between several distinct services (PaperMC server, Bungeecord relay, Discord server, and CowBot, which communicated between them all), and this fact made testing in any manner other than manual end-to-end very difficult. As an aside, I did consider mocking these services for some unit tests, but determined it had low reward -- the changes I made were unlikely to be broken by future patches. This is all to say that the CowBot repo (prior to my contributions) has no integration testing. Several PCN repositories *do* have CI/CD processes, but CowBot is not currently one of them.

That said, any potential bugs which slip through this lack of integration testing will most certainly have *low severity* -- at worst, CowBot itself (which runs in its own thread) would crash, and the services which it provides (message relaying, Discord:Minecraft account linking) would simply no longer be accessible.

Developer time is expensive, and if I were a PM for PCN, I would make the same choice: implementing a testing framework for CowBot is probably not a good use of time while your team is still small, especially if you're only making small changes which you can manually verify.

For task 1 and the first part of task 2, then, I stuck to manual testing. As additional QA (for all tasks), I aided the natural GitHub PR passaround code review process by adding GitHub comments with "why" justification in relevant parts of my code (here, here, here). These changes truly are simple, and for the reasons outlined in the above paragraphs I believe this level of QA to be sufficient.

For the second half of task 2, however, my contributions were quite large and complex, and manual testing alone would not be to my standards. (I also doubted the maintainers would accept such a large change without tests). Once I believed I had the parsing and translating code working (via manual tests on my local development server), I added a JUnit test dependency to CowBot along with a unit test suite for the parsing module I'd created, the DiscordToMinecraftFormattingTranslator. I initialized the suite with short, meaningful tests, then used JaCoCo (Java Code Coverage) to generate its line and branch coverage metrics. I then added smaller tests to fill the gaps in coverage and re-generated JaCoCo reports until I had a suite of 20 tests with 100% branch and line coverage.

I chose unit tests as my main source of QA for this task because my parsing and translating module had its responsibilities well divided, and thus the input & output of the parser could be easily verified. In my implementation, I represented a formatted message with a container of [start, end] pairs for each type of formatting. For the case "*message*", it is easy to see that the italics start at index 0, and end at index 8. More importantly, it is easy to assert with unit tests.

EECS 481 HW6b
Michael Jecmen
mjjecmen

PLAN UPDATES

ACTION	ACTUAL HOURS	PLANNED HOURS	EXPLANATION
Create test Discord server	4.0	2.0	This was just way more work than I anticipated. There was no formally documented process for setting up a test PCN environment, and even when I asked for guidance there were a few things which the devs forgot to mention (namely missing external dependencies and plugins which were required in multiple places).
Task 1 implementation and testing	1.0	1.0	Once I was able to reproduce the bug, the fix itself was trivial. All of my manual tests passed on the first go.
Task 1 PR write-up	1.0	2.0	Most of the time here was spent screenshotting some of the results I had found through manual testing, and reverting the changes to show what the previous implementation yielded for those tests. The write-up was easy: there weren't many changes to comment on.
Task 2 implementation and testing	14.5	2.0	1.5 hours for the color formatting bugfix. 13 hours for the text formatting feature. Of that: - 1 hour researching MC and Discord formatting. - 3.5 hours determining I can/should not use existing Markdown parsing code. - 5.5 hours of implementation until my manual tests passed. Turns out writing a parser is *really hard*. - 3 hours testing. Unit testing was refreshingly easy to integrate with Maven's build process. Even though I spent a *long* time on this task, I am glad I didn't try to use existing parsing code -- my time spent researching makes me think it would've been a dead end. It's either a rare case of it being better to build an in-house solution, or I've fallen for the exact ego trap outlined in lecture.
Task 2 PR write-up	1.0	0.5	Because of the sheer volume of code I committed to this PR, I spent a bit more time than I anticipated adding comments and marketing my contributions.
Create test MySQL server	2.0	1.5	Along the way I ended up thinking the MySQL test server was necessary for Task 2, even though it wasn't. I set it up only to realize my error had been with another module. The time discrepancy was from me learning how to use MariaDB.
Tasks 3 and 4	0.0	6.0	Did not start. As you can tell, I only got to half of the tasks I allocated for myself in HW6a.

TOTALS	23.5	15.0	*Always leave room for improvement.*

EXPERIENCES AND RECOMMENDATIONS

General Thoughts

If I were to re-build PCN from the ground up, there really isn't much I would change in terms of infrastructure. The developers clearly know their DevOps stuff. My only real complaint is the lack of documentation in each repository -- in my experience, high quality projects have extensive documentation (architecture diagrams, code comments, READMEs, etc), and especially for a group that is trying to attract more open source contributions these forms of "onboarding" seem vital. Despite this, PCN's code readability is still quite high due to the modularity of their code -- classes have clear, concise purposes (even if those purposes aren't explicitly documented), and are largely decoupled from each other.

A more nuanced, positive take is that PCN certainly took the right approach by avoiding using multiple languages wherever possible. I hate programming in Java as much as the next engineer, but when Minecraft and all of its plugins are written in Java, and you're going to write a Discord bot to interface with said Minecraft plugins, the best choice of language for ease of integration and extensibility is Java. I've worked extensively with a WebAssembly multi-language project (Typescript and C++ interfacing with each other -- look up the Emscripten compiler) and that "glue layer" is a constant source of headaches for me and my team. So, although the actual programming which goes on inside of CowBot would certainly be easier in Python (*why can't you index into an ArrayList with []???*), avoiding that glue is certainly the right choice for PCN.

The most difficult part of this process was understanding the existing codebase and how all of the services worked together. There were hours I spent during setup thinking I'd have to choose another project -- periods of build error after runtime exception after plugin crash where I genuinely didn't know what to do next, or if any of my fixes were even making things any better. This is very consistent with my experiences in industry: build errors for large software engineering projects (especially when you're first getting set up on a company machine) are usually *extremely* difficult to correct without help. The 4 additional hours of setup beyond what I spent in HW6a would have been 40 if I hadn't been so loose lipped with my questions.

On the Social Nature of Open Source or Team-Based Software Engineering

The most valuable lesson I've learned during my experiences in industry (and applied in all of my work since) is easily that: ask questions. It's not a flashy lesson, but it must be crazy how many people don't do this -- at my previous internship (which is still my current workplace; I've continued contracting for this company since the summer) I was often praised for not being afraid to ask questions. When entry level engineers join existing teams at large software companies, it's difficult to avoid the feeling of impostor syndrome. Everyone's smarter than you, already knows the codebase, and are all (usually) friends with each other. You've got none of that, but when you're faced with confusion or feel lost in any way, the alternative to seeking help is to *simply take longer on your task*.

Nobody knows what they're doing when they join a new organization. It's obvious, but once you truly recognize that, the hesitation you have in asking questions will fade away. To relate this to once more course concept before I step off my soapbox -- in my experience, interviewers *love* to hear this. When asked to name a time I learned something, I always use that answer -- recruiters eat it up. Needless to say, throughout this process I constantly leaned on my communication with the PCN devs to keep my hours productive. I wasn't given any explicit feedback on the level of collaboration I had with (imposed on) the team, but I'm sure the relatively small time spent answering my simple questions was an acceptable cost for my contributions. I think this would be a great topic to touch on during the productivity lecture -- perhaps it's obvious to senior engineers, but question asking / getting yourself "unblocked" is a critical soft skill in software engineering that this course doesn't cover.

Testing

I would definitely recommend JUnit and JaCoCo, and was thoroughly impressed by their seamless integrations with Maven. Setup for each of these dependencies took so little time (they're so widely used I could Google "how to set up _ with Maven" and get correct configuration information) that I was writing unit tests within 20 minutes (doing the initial research to determine that these were the tools to use took longer). I took a HW1a-style approach to testing here: I manually wrote tests for all of the relevant cases I could think of, then ran JaCoCo to see which lines and branches I'd missed.

Perhaps some notion of Cyclomatic Complexity can be a decent maintainability metric after all -- flat code *is* certainly easier to test than deeply nested code. I was taught that flat code is more maintainable and readable than nested code in EECS 381, but I'd never considered how easier code coverage could result from that. It didn't take a constraint solver to come up with the path predicates I needed to cover the gaps my initial test suite left in coverage -- I could come up with them in my head and effortlessly generate the remaining tests. For this reason I suspect flat code would be more easy for static analysis tools to run on as well -- easier predicates means faster results and less false positives.

Most surprising about the testing process was the fact that the devs actually asked me to set up VSCode integration for the tests I wrote, and commit an updated .vscode directory to the repo. Every piece of advice I've ever seen about git tracking and VSCode development is that the .vscode folder should be per-environment state (i.e. it should always be excluded from commits). I suppose this is one benefit of a small operation, however -- you can know for sure the set of machines your code will be running on, and can manually verify the build and integration proces works on them with ease. As a disclaimer, I may or may not get to this additional ask before this is graded (the comment is from 3 days before the deadline, and I've already written all other sections of this report). Here's the ask:

> *Can you add tooling to .vscode/ so that VSCode recognizes there are tests and can run them? This repo is a bit behind in our tooling, but the template has extension suggestion. I am not sure what all is needed after that to get the test runner to recognize the test cases.*
>
> *Otherwise this looks good and we appreciate the time and thoughtful approach you've put into this :)*

I'll certainly implement this, but it may be after this report is graded. In any sense, I was quite surprised to see the .vscode folder not present in the .gitignore, and more surprised to see a specific ask for it to be updated. I guess that further goes to show that real software engineering is about so much more than just code: build systems and testing frameworks matter *a lot*. I've learned more and more over this past year that these skills are *highly* in demand.

Regrets and Takeaways

I *really* wanted to get fuzzing working on my parsing code for task 2. I knew it was a gap in my QA, because on average, complex string parsing code has a pretty high chance to fault. I thus thought this was a golden opportunity to inject one of my favorite course concepts -- fuzzing. I tried to use AFL and Kelinci (I followed this article's tutorial) to fuzz my parsing module, but ultimately gave up after I spent too long on it without a sign that I'd ever get it working. I justified leaving the translator un-fuzzed twofold. 1: I did not have time for it -- I was already way over my time budget and (unfortunately) I have other responsibilities in life outside of EECS 481. 2: The possibility of CowBot being killed by a malicious Minecraft chatter has *low severity*. As mentioned in previous sections, if CowBot dies, no critical services die with it. The largely aesthetic benefits it provides can simply be restored whenever a dev notices and restarts CowBot.

This may be slightly contrary to course recommendations, but it was extremely rewarding to contribute to a small project, and I highly recommend it (provided it's active). The developers clearly appreciated my time and effort, and were invested in my success. I never had any communication issues with the core developers, which is more than I can say for an entire internship I had two summers ago. It would sometimes take days for my manager to respond to basic questions, and he was being *paid* to train and develop my skills. On the contrary, at least one of the PCN developers was *always* online and willing to answer questions.

Furthermore, I am very glad I chose a project with an active surrounding community. As shallow as it sounds, it was such a great feeling to see my contributions valued by *real end users*. Even from my very minor change which was accepted I saw discussion from PCN members in the Discord server who had noticed and commented on the bug fix. It's probably quite rare (outside of dedicated QA positions) for you to observe such unfiltered feedback from end users when your code doesn't make a major change, and I recommend it -- it's fulfilling! To tie this into a course concept, Dr. Ciera Jaspan discussed developer burnout being a challenge of extreme importance for software companies. I'm no expert, but I'd be willing to bet there's research out there about positive feedback for a job well done and developer satisfaction!

EECS 481 HW6b
Michael Jecmen
mjjecmen

ADVICE FOR FUTURE STUDENTS

In any work-related context, ask for help unapologetically and as often as you need to.

(I am totally cool with this report being made public for future students.)

EXTRA CREDIT

My first accepted open source pull request!