Summer of Nix 2021

A Program Report

Matthias Meschede

The Summer of Nix 2021 was a large coordinated effort to reproducibly package Open Source software, making it readily available and usable by anyone. The program brought together independent developers and community enthusiasts for two months to write build instructions with the next-generation reproducibility-first package manager Nix. This report retells the story of how and why the project came into being, how it got shaped by constraints and design choices, how it was organized, what problems it encountered, and finally what its measurable outcome was, as well as feedback from the participants. We hope it is informative to past and future funding organizations, organizers, and participants.

1 Context

1.1 Free and Open Source Software in a complexity crisis

The Internet should be an open, decentralized, and privacy-respecting network of peer nodes, built on top of an ecosystem of Free and Open Source Software (FOSS), allowing anyone to build new and innovative applications, oriented towards human needs and respecting individuals' privacy. This vision is pursued by many software community and political actors,[1] and ultimately derives from the fundamental idea of a participative society of sovereign citizens. It contrasts with present day reality, where operation and development of technologies and standards related to computer networks are dominated by a few global actors, each with incentives and history of pushing adoption of their own, closed products, services, or software ecosystems.

Unfortunately, an ecosystem of Free and Open Software comes with complexity that tightly controlled proprietary environments can often avoid. In practise this means that FOSS applications are often more difficult to install, configure, and run than proprietary alternatives — a natural consequence of the enormous diversity of approaches that thousands of individual developers without centralized organization come up with. These developers use their favorite programming languages, software libraries, operating systems, distribution formats and platforms. Their software most of the time needs to be compiled from source, a custom originally intended to facilitate portability, as opposed to distributing binaries which are tied to a specific operating system or processor architecture. Although this diversity is a *feature* of an open ecosystem, greatly fostering innovation and resilience, it comes at the price that its software is often more difficult to actually set up and run. The cost of adopting a set of FOSS applications, is thus often higher than paying for an out-of-the box proprietary, fully

[1] European Commission, "The European Commission's Open Source Strategy for 2020-2023," n.d., https://ec.europa.eu/info/sites/default/files/en_ec_open_source_strategy_2020-2023.pdf.

integrated, but opaque solution. This problem has been recognized for a long time in computer science circles.[2]

The package manager Nix[3] is an emerging technical solution to manage FOSS complexity, and as such, a central component to make the Open Source vision a success. Making FOSS easily available via Nix was the "raison d'être" of the *Summer of Nix* program.

1.2 Institutions and their interest in the Summer of Nix

Four institutions with different interests teamed up to realize the *Summer of Nix*:

The **European Commission**[4] (EC) was the initial *source of funding* for Summer of Nix via their Next Generation Internet (NGI) initiative. The goal of NGI is to "shape the development and evolution of the Internet into an Internet of Humans. An Internet that responds to people's fundamental needs, including trust, security, and inclusion, while reflecting the values and the norms all citizens enjoy in Europe."[5] NGI is funds a large variety of projects and subprograms with 312 million Euro between 2018 and 2022.

The **NLNet Foundation**[6] (NLNet) is an independent foundation, largely aligned with the above stated mission of the EC's NGI initiative, funding independent developers to work on FOSS for a long time. NLNet had been charged by the EC, via their cascading grant mechanism, to *distribute* parts of the NGI funds. This specifically concerns funds from a subprogram called NGI Zero[7] that "provides grants to individual researchers and developers as well as small teams to work on important new ideas and technologies that contribute to the establishment of the Next Generation Internet." Within NGI Zero another sub-initiative called PET[8] focuses on privacy and trust enhancing technology, which ultimately became the principal source of funding of Summer of Nix. Plenty of FOSS solutions, which NLNet funded within this setting, faced the outlined FOSS complexity challenge to reliably build and deploy. Looking out for a technical solution, NLNet chose Nix as preferred packaging strategy because of its unique reproducibility guarantees and other features such as composability that allow to realize their and NGI's wider vision. This is how parts of the funds became available to package NGI-funded software with Nix. NLNet has approached the NixOS Foundation to help them with this packaging effort.

The **NixOS Foundation**'s[9] mission is "to support the infrastructure and development of the NixOS project as a whole." It's scope of action is limited to essential tasks in the Nix ecosystem due to limited resources, but also because Nix is managed to a large extent by a vibrant and self-organizing community.

[2] YouTube, "Linus Torvalds on Why Desktop Linux Sucks," n.d., https://youtu.be/Pzl1B7nB9Kc; Roberto Di Cosmo et al., "The Mancoosi Project," n.d., https://www.mancoosi.org/.

[3] "Nix: A Safe and Policy-Free System for Software Deployment." n.d.; Nixos Community, "NixOS Website," n.d., https://nixos.org/guides/how-nix-works.html.

[4] European Commission, "What the European Commission Does," n.d., https://ec.europa.eu/info/about-european-commission/what-european-commission-does_en.

[5] The Next Generation Internet website, "About the NGI Initiative," n.d., https://www.ngi.eu/about/.

[6] NLNet, "Foundation Website," n.d., https://nlnet.nl/.

[7] The Next Generation Internet website, "About Ngi0," n.d., https://www.ngi.eu/ngi-projects/ngi-zero/.

[8] The NLNet website, "About PET," n.d., https://nlnet.nl/PET/.

[9] NixOS Foundation, "Mission Statement," n.d., https://github.com/NixOS/nixos-foundation.

The availability of funding for Nix packaging from NLNet was a welcome opportunity for them to grow and improve the Nix ecosystem. The NixOS Foundation thus started to work on packaging with individual contractors. They set up infrastructure such as a continuous integration server, and a task list, before the idea of Summer of Nix even came up. But the NixOS Foundation lacked the manpower to drive this program forward more actively. That is how the idea of a collaboration with Tweag emerged. In this collaboration, the NixOS Foundation was the *administrating organization* of the Summer of Nix, handling contracts, payments, IT infrastructure and more, but could largely stay out of program design and daily management.

The final actor, **Tweag**,[10] is a software consultancy and one of the principal enterprise user and contributor to Nix. Tweag sponsored developer time for project management, to actually *design, organize and run* the Summer of Nix. Tweag's interests are to improve and give visibility to Nix, and to foster and participate in the lively community around it.

2 Designing and running Summer of Nix

This section is written for those interested in the organization behind Summer of Nix, the program constraints, the reasons why certain choices were made, and also what organization of such a program entailed concretely. If you are interested in the outcome only, you can skip to Contributions.

2.1 The idea emerges

The initial idea to run a large, concentrated community program in the summer, developed as a reaction to feedback from the Nix community about the already existing packaging effort with individual developers working independently through the numerous NLNet projects. The existing effort was not advancing as quickly as intended, and did not appear appealing enough to attract enough applications.

Many community members felt unqualified to work professionally with Nix, because, as an emerging technology, it often had been only a hobby for them. Many also felt that there might not be enough support and time to learn the required skills on the fly during professional work. The entry barrier to applying for a packaging job was therefore perceived as high.

Another aspect was that independently working on packaging was in itself not regarded attractive work, where besides delivering one could have learned new things and made interesting connections. This was reinforced by the fact that many of the NLNet tools to package were still experimental, in prototype or alpha stage, without significant user base, and some having no regular maintenance. Finally, although the program was appropriately funded, it was not as financially attractive as some software industry jobs.

The main challenge therefore was to design a program that was more attractive and fulfilling, taking into account the *non-monetary* interests of potential participants as much as those of the funding organizations.

Some of the program's principles emerged naturally from these considerations. Participants should explicitly have time to **work, learn, and meet**. Coding

[10] Tweag, "Company Website," n.d., https://tweag.io.

should not happen in isolation but in groups, with regular discussions and feedback about the work. Participants should not only learn but also teach their peers to make use of their skills as far as possible. The program thus became much more than a simple programming task, and we hoped that it would be much more attractive for participants.

However, the challenge was to actually *realize* a project in the above spirit, open for beginners, with freedom for each participant to learn, teach, experiment, and meet while applying their creativity, ideas and newly generated knowledge to broadly useful work. One implication of this was that we had to part with a professional software engineer's salary to stay within our given budget and to deliver the expected outcome with the allocated money. Instead we decided to set payments slightly above Google's Summer of Code[11] rates.

Another challenge was that we had only a very limited number of organizers available to actually run the program: Even I, as the principal organizer, was by far not able to work full time on this project, although his employer Tweag generously supported him as they could. In addition, a small group of largely volunteer organizers helped off and on, depending on their time. It was clear from the beginning that a program with an emphasis on community and team work required much more work and that we needed further help, possibly in the form of a few selected people who would be experienced with Nix and willing to coordinate teams. The idea of "mentors" was born.

The limited amount of dedicated time on the organizational side was probably the most constraining factor throughout the program, more than the budget. This was further exacerbated by the fact that we were starting out from scratch, without prior experience or infrastructure. This constraint was clear to everyone from the beginning, but the opportunity to realize the program was given, and we were convinced we had a good chance to deliver decent results despite the constraints.

2.2 Fixing the basics: compensation and time frame

The general idea "try running a condensed community program to make NGI packaging more fun, effective and valuable for everyone" was thus clear, but what was *not* clear was the number of applicants to expect for something like this, and also not how to put it in practise. Initially we estimated to receive 10-15 applications, of which certainly some would drop out. Running the program with a team of 5-10 people seemed realistic. With this in mind, we fixed compensation, the time frame from August 2nd to October 15th 2021, and defined the mentor role, such that we could make a public announcement.

Compensation was set to a flat base rate of 2750 Euro per participant for 8 weeks of full time work anywhere within the EU — a gesture to the entity who ultimately funded this project — and for other locations adjusted to purchasing power, based on a list from Google's Summer of Code in lack of a better resource. We considered worldwide equal pay as well. Arguments for and against were discussed, but ultimately we decided for adjustment, mainly because this was recommended by the European Commission and others who had more experience than us with the implications of one over the other. It meant that participants from EU countries with lower purchasing power received the most attractive offering due of the flat rate, which corresponded to the conditions at NLNet Foundation's seat in the Netherlands. Worldwide payments were handled by the NixOS Foundation and NLNet, experienced with this type

[11] Google, "Google Summer of Code," n.d., https://summerofcode.withgoogle.com/.

of situation. One additional thought about compensation was that we did not determine it by simply dividing the total available budget to spend it all, but based on a pre-existing expectation of what should be delivered for the money. With the above compensation, a participant, on average would have to deliver roughly three packages over the eight-week period — a number that we expected to vary significantly, as the software to be packaged was so diverse. Still this number remained our guideline to see how this most freely organized community event faired economically in direct comparison with independently working developers, not taking into account the additional benefits of a community event for the developers and the community besides packaging.

Finding a time frame turned out difficult. Choosing August 2nd to October 15th 2021, we tried to target university breaks across a variety of countries. We did not have the data to base this decision on, because we did not know where participants would be applying from, but also because it is not easy to obtain. Each country, and often each university within a country, has different season schedules. The southern hemisphere was, as unfortunately too often, somewhat disadvantaged by the fact that the program coincided with their winter break that is often shorter than the summer equivalent, and already by the name of the event. We were not able to find a better solution for everyone within the limited time we had. For participants, we designed the program for 8 weeks of full time work (320 hours) in the total 10-11 week time span, similar to Google Summer of Code. Targeted results could be accomplished while leaving flexibility to take vacation or start earlier or later. We would not grant too much flexibility to make sure that there was enough overlap for everyone to actually work together in a team. The compensation stipend was to be paid in two stages, after working the first 160 hours and then the second 160 hours. When we came up with this, we expected to see tangible outcomes in the form of packages and other code contributions in the second half of the program, and expected the first 4 weeks to be required to get up to speed. This turned out not to be true at all.

The mentor arrangement was different. For the mentors, we kept the original rate received by independent NLNet packagers, about 50 Euro/hour — significantly higher, and with corresponding expectations and responsibility. This meant that we could not possibly employ mentors full time. We thought mentors could be present for about 8 hours per week to coordinate and answer questions, and deliver a total of 100 hours over the 10-11 week program.

2.3 Announcement and interviews

These basic decisions were enough to announce the program on the NixOS Discourse[12] and await applications. We did not want to shape up the full program before knowing who would actually participate. Applications were handled over email as we tried to avoid sign-ups and reduce the barrier to participation as much as possible. Additionally we deemed email more personal than forms, especially when expecting maximum 15 applications anyway. We also decided to have short 15-30 minute interviews with every applicant to get a personal connection and also to answer potential questions.

The response to this announcement was overwhelming. In total, we received **94 participant and 10 mentor applications**. Given the high number of applications, email quickly became a liability, although we partially automated it

[12] The NixOS discourse, "The Summer of Nix—Learn Nix While Doing Useful, Paid Work," n.d., https://discourse.nixos.org/t/the-summer-of-nix-learn-nix-while-doing-useful-paid-work/12225.

by drafting personalized messages directly from a central sheet. Getting applications and responses to these messages in a more structured form might have simplified things, but we were now stuck with this decision.

In the application email template, we asked a few simple questions:

- Who are you?
- Why are you interested in this program?
- What are you able to do with Nix?
- What would you like to learn?
- When are you available?
- What time zone?
- Specific requirements?
- Optional CV

These questions turned out to be very useful to quickly assess the applicant's goals. Following our initial decision, we did over 100 interviews in about one month with a team of four volunteers, who were conducting them besides their paid work. In hindsight this was probably not worth it and quite exhausting, but it also gave the program a very personal, respectful and warm touch. As interviewers we got a much better feeling for the Nix community, the wishes and goals of the applying participants, and therefore ultimately the program that we were going to run. During these interviews it became clear that we had many enthusiastic, extremely nice, and very knowledgeable people applying.

2.4 Total number of seats, selection process and team building

The sheer amount and the high quality of the applications was extremely motivating, but it also put us in the unpleasant situation of having to select and pick one good applicant over another — something we had hoped would not be necessary.

Ultimately we had a limited number seats in the program because of our budget but its upper limit was quite generous, more than what we could and wanted to spend in a single untested program. The decisive question was therefore whether we could make effective use of the money that we planned to spend and how much we needed to retain to cover the risk of falling behind the expected work. We had the choice to run a small directed program, or try running a slightly bigger one with an autonomous team structure that could scale. We decided to go with the latter for various reasons:

We thought a program that scales, with autonomous teams, was necessary in any case, as we did not have the resources to conduct top-down direction. It also seemed more enticing and in the spirit of the program as a community event to allow for autonomy and decentralization, with opportunities to participate, network, and connect.

The final number of program seats that we came up with balanced out a variety of considerations: We wanted and needed to move faster than with the previous packaging effort. Spending budget means, for funding organizations, not only to invest optimally in terms of result quality, but to do so timely because funding is available for a limited amount of time. This time factor is often neglected. We were not under immediate budget pressure of having to spend the whole amount, but we needed to speed up spending to stay on track with the goals of the funding organizations (ultimately the EC's NGI program). But why not spend it all if the opportunity is good? Would there be a better one later? We

went for a compromise and decided to invest slightly more than half of what the budget allowed, retaining a significant part to hedge against the risk of failure, and also to have money for potential follow-up work.

With all of these reflections, and given the number of applicants, we decided to try building **seven teams that each had five developers and one mentor** as basic team structure. This meant a total of **35 participants and 7 mentors**. Based on the applications, we decided to add one extra mentor who would work on cross-team tasks, which will be described later.

Then we needed to fill these teams, and, since this program was *not* about just delivering code as quick as possible, but also about learning together and meeting other like-minded people, we did not want to simply prioritize by experience with Nix. In contrast, team **diversity** seemed to be highly desirable to actually make a fun program in the "work-learn-meet" spirit outlined above, naturally creating opportunities to learn from and have interesting conversations with each other.

Unfortunately team diversity in all aspects was not easily achievable: A first decision that we felt forced to make was to have timezone-homogeneous teams that were not spread over more than **4 hours of time zone difference**, ideally less. Meetings and synchronous communication, an essential part of the team idea, seemed too hard to organize with more than three people spread over America, Europe, and Asia. We thus focused on geographical diversity *across*, but not within, teams. This lead us to reserve 15 spots (3 teams) for Central European/African time zones, 10 spots (2 teams) for American time zones and 5 spots (1 team) for East European, Middle East and 5 spots (1 team) Eastern Asian time zone. There were no applicants from other parts of the world. Although this distribution is biased toward European/African time zones, to be expected with EU funding, this choice of numbers was mostly imposed by the number of applications from each region, as by far most of them came from Europe. Paradoxically it still gave a significantly higher chance to non-Europeans to participate, although there were less overall slots for them. With "non-European" here we strictly refer to place of living at the time of the program, and some of those non-Europeans were Expat-Europeans. Therefore, although a range of continents and cultures were represented, participation was by far not globally representative, not even if restricted to highly industrialized nations.

Another shortcoming, as unfortunately so often in the tech world, was in failing to achieve gender diversity. The only two (very strong) female applications were not enough to even consider forming somewhat gender diverse teams.

All this meant that we were left with building teams that were diverse along the axes of **professional seniority** and **Nix experience**. We did this in a way which we called, for the lack of a better term, compartmentalized randomness: building on a 3×3 matrix of seniority and Nix skill a random queue of applicants. The process was certainly not objective, just as basically any application process, but we are convinced we were guided by reasonable principles. It meant in the end that senior Nix experts did have the highest chance to participate, simply because there were not as many of them, but it also gave a fair chance to others.

We could have considered additional axes to build the teams, for example, by grouping people who were interested in similar technology. But this seemed to become too complicated under the other constraints and also premature because we did not go through the actual work yet in detail and had not thought about how to actually distribute it among the teams. Perhaps, rather than searching for ways to further optimize teams upfront, an additional mechanism

to adapt and improve them during the program would have been useful — but also potentially dangerous if done maladroit.

2.5 Timeline, task distribution, events and start

Everything that has been described so far, from the first idea to the end of the selection process, happened in a relatively short time period: It began with a first mention of the idea in an email on January 22nd 2021. A direct follow-up, one of the oldest of about 500 Summer of Nix related email threads (excluding auto-generated notifications) from February 6th 2021, reads:

> we could have 4 participants FT for 5 weeks including budget for the mentors. This is probably the most challenging but also most rewarding endeavor and would need to be organized quickly if it is for this summer.

The NixOS discourse announcement post came out on March 21st 2021, and we reached the end of the selection process on May 16th 2021 after more than 100 applications. There was not much of time for planning afterwards as well, because after about one month of downtime, we had to start preparing work and events for 35 developers and 8 mentors to avoid major confusion on August 2nd 2021, when the program was supposed to start.

As first preparatory measure, we started regular meetings with the mentors to on-board them gradually to the program so that they would know what was coming and had the means to already start influencing it. The first question to answer for all of us was what we would be working on exactly and how work would be distributed:

Thanks to prior work from the NLNet and NixOS Foundations, we already had a written list of relatively independent work items, which essentially amounted to the more than 200 NLNet projects to package, prepared as a long list of GitHub issues. Certainly, some information was missing in those issues, there were duplicates, and some of them were simply not tractable, but overall this list was a treasure trove, and we would have had trouble to get ready in time without it.

Initially we thought about going through the issue list upfront, tag and distribute it to the teams, e.g. grouped by technology. We quickly realized that we would not have the time to do this before start of the program, and that the list was too large to deal with sequentially. We therefore opted for a different method which I will call **assign-and-trade** here: The full issue list was randomly divided by the number of teams (seven), assigning every issue a team label. Coincidentally the full list had precisely seven pages on GitHub which means that splitting and labelling was easy to do. This distribution was definitely not optimal, but rather a random starting point: It meant that some teams would have more difficult issues than others, some teams would be missing the required skills to tackle an assigned issue, or they simply would not be interested in it. This is why we introduced a trading mechanism, such that teams could report interest in an issue they liked, or put issues out for trade which they did not want to work on.

As further preparation, we also wrote a step-by-step packaging work flow to set some standards to get everyone started quickly:

As developer you look through the issues assigned to your team via filtering by GitHub label. Once you find an interesting one, you assign yourself to it. An

issue corresponds roughly to an independent NLNet project. The first step after assignment is to find out what it is about exactly, and what could actually be packaged. Once this is done, you start packaging by forking the original repository to the NGI-Nix GitHub organization, adding the Nix build instructions directly to it. Once ready, there are several paths to continue: a PR to the original source repository (that is why we initially thought that forking was a good idea), a PR to `nixpkgs`, Nix's official package repository, or simply leaving it as independent build instructions on the `ngi-nix` organization in an independent repository.

Participants would work through this workflow alone, and ask their mentor or other team members for help and support.

A question that came up was when those build instructions were considered done. As with most software work, one can deliver more or less complete and high quality solutions depending on the goals. We worried a little about quality in general, because often developers are satisfied if something works for *themselves*, but not necessarily for somebody else. Although Nix gives certain guarantees about reproducibility, it cannot guarantee that something is complete and documented enough so that it is usable. To drive quality up on the finish line of packaging, we decided to ask the 8th mentor to explicitly focus on this, to test the packages, give feedback and try to raise the bar. We hoped that this would allow the other mentors to concentrate on working with their team in a supporting role as much as possible, without having to switch too much into an adversarial position themselves.

With the issue assignment strategy and the basic work flow, we felt ready to get started, but we were also scared to actually do so. Having 43 participants and mentors all starting out at once without any test seemed audacious. The fear was not that our upfront preparation and game plan wouldn't work out perfectly. It was always thought more as a guideline for self-thinking individuals. With the little time that we had to organize the program we did not hope to get anywhere close to perfection. But we feared to have missed something major that would block and demotivate the brightest participants right from the beginning. Fortunately, two participants and their team mentor were willing to get started already in July 2021, one month before the actual start date on August 2nd, to try what we had prepared. They began to package the first repositories, started to flag up problems, and got used to the program.

Another aspect not mentioned so far was our plan to organize a presentation schedule for Summer of Nix — inviting core members of the community to talk about what they know best: their own work. Due to time constraints we were not able to organize this much in advance, and decided to go for a more interactive and flexible approach. We thus only had prepared up front a kick-off meeting on Monday, 2nd August 2021, and the first technical presentation by the inventor of Nix, Eelco Dolstra, and later added others on the way through the program.

2.6 Tooling

During the preparation phase, we had to make decisions concerning which tools we should use to organize this event. **Simplicity** over fanciness was our major guideline here. In addition, we tried to use **FOSS tools**, because it corresponded to the overall spirit of the program, and because they were often quicker and easier to use, as no further budget and sign-ups were required. We also decided to be **pragmatic** and use commercial tools whenever they were providing us with a clearly simpler and more robust solution.

Some of the FOSS tools we used (none of them self-hosted) were:

- **Cryptpad** for collaborative text documents and sheets. Cryptpad was quite feature complete and a good choice for collaborative writing and sheet calculations for organization. In our case, it competed with Google Docs, which was probably better known by many and directly integrated with Google Drive. The latter fact was definitely an advantage for us, which is why we did not fully move to Cryptpad.
- **BigBlueButton** for webinar-style presentations. We used an instance hosted by the TU Delft for this event to great effect. Other hosted instances were difficult to find. Especially the shared-notes and recording features of BigBlueButton were helpful to make the presentations interactive.
- **Jitsi** for face-to-face video conferencing worked very well. The fact that there was no requirement sign up anywhere was a big plus, especially to organize the hiring event later on.
- **Matrix** for synchronous chat also worked well. The user experience of its main client Element (formerly Riot) was sometimes not as polished as in tools like Discord or Slack, but in general it did what it was supposed to do. The "Space" feature of Matrix, basically a set of channels, was very convenient to integrate open community channels with our private event channels. We had a very simple setup with two general chat rooms, a single private one for each team, and in addition an admin and mentor channel.

Coincidentally, all of these open source tools had received NLNet funding, and we worked on improving packaging for them during the program. Some of the commercial tools we used in addition were:

- **Hellosign** for signatures
- **GitHub** for the issue list, program documentation, discussion board, and source code
- **Google Drive** and **Google Docs** to store administrative documents

During the project, we also worked on FOSS alternatives to these, so that in principle we could have run the full program on FOSS. However, as mentioned earlier, we had limited resources to set this up, and needed to carefully budget what to spend our time on. In addition to these tools, we mainly used email for administrative purposes, for example to send out `.ics` calendar invitations and other cases where we could have used an external tool. This was advantageous in a way, as everyone knows email, and because it was possible to automate it to some degree — but given the size of the project more structured approaches could have helped.

2.7 The program runs

The strange thing about bigger projects from the perspective of the organizer is that the start does not really feel like the start, because so much of the work has already happened. And as soon as the project starts, it feels, like a boulder rolling down the hill, hard to influence where it will end up. It paid out to have clear, built-in and scalable day-to-day mechanisms for feedback, guidance, and decision-making, via the mentors who were admittedly dropped into a situation with significant responsibility, little time for preparation, and tight time budget, officially being only available one day per week for coordination. The mentors handled the major part of the organizational work load from day to day; guiding, organizing and resolving as much as possible directly with their teams. The bottlenecks that we did encounter were the decisions that could not be made by

the mentors, or where it was not clear if they had the authority to make them. In this case, questions were handed upwards, mostly to NLNet for whom this was a significant amount of work to stay on track. During the program we had little problems with bad decisions, but with bottlenecks and a lack of responsiveness when decisions went upwards, which was sometimes frustrating and a sign that local decision making could have been even better, and surely something to improve on.

In this decentralized situation no one had an oversight of the whole program. Work was distributed and trust-based, and everyone only had their own limited perspective. Participants knew most about the packages they were working on, mentors had a larger vision of their team activities and also the issue list, organizers were busy organizing, and NLNet and others busy resolving questions. We did not have effective and fast reporting mechanisms in place. Our global issue list captured only a small part of what was going on. In addition, participants filled out time sheets with weekly granularity but we did not have the time to read those until after the event. Weekly granularity turned out to be an excellent choice though because we were at least able to read all of them later. Other than that, organizers and mentors exchanged informally on progress in weekly meetings and via Matrix.

During the program, participants were mostly working on packages, learning, and talking to each other autonomously. Organized events set a certain rhythm for the participants throughout the week: typically a team meeting (although this was left to each team to decide), and about every second week on Wednesday at least one centrally organized presentation, in two editions to cover all time zones. These presentations were given by invited speakers, some of them program participants themselves, about in-depth but basic Nix topics rather than about the newest developments.[13] We also had a time slot on Tuesdays for participant presentations, which was used occasionally. More happened behind the scenes, proposed by various participants, and in the different teams. Later in the program we sent out a list, randomly assigning one or two meetings with someone from another team every week to foster cross-team communication.

The program ran in this manner more or less autonomously. I, as main project organizer, was even able to hand over coordination for two weeks to go on a vacation after a very intense half-year. The program went on without notable accidents, although of course smaller problems came up. As was to be expected, many things could be improved *even under the same program constraints* if we were to repeat this program. But some of what exactly could be improved only became clear *after* the event, when we asked for detailed feedback, which is covered in a Feedback section of this report.

2.8 The hiring event

Before the program started, another idea materialized that seemed very valuable to pursue, even at the expense of having less time for organization of the main program:

As consultants at Tweag we often talk to companies using and adopting Nix. Similar to other emerging technologies, Nix is often a blessing and curse at the same time for them. As an emerging technology, it solves previously unsolved problems, but it also does not come with a vibrant job market. Immediately after

[13] Subjects included Nix Flakes, Nix modules, the Nix RFC process, the `nixpkgs` release cycle, `poetry2nix`, `dream2nix`, EU Open Source politics.

the interview phase, we concluded that companies may have great interest in meeting the candidates who applied for Summer of Nix, and that the same may be true for many applicants who often considered Nix not as something they could find a job with. The idea to connect Summer of Nix applicants (even those who did not get seats on the final program) with companies adopting Nix felt natural, but it meant further organization at a moment when we had little capacity.

Still, we took the decision that this was worth pursuing, simply because it felt like a hiring event would complete the Summer of Nix: It provided a great point of motivation and direction, and made it clear that, from the junior participant perspective, Summer of Nix would be about giving talented community members an opportunity and guidance to make the jump into professional life, and to gain experience in delivering paid work in a protected environment. From the senior participant and mentor perspectives it was a moment to make use of their knowledge, teach, and finally connect with companies who might be interested in working with them professionally. However, the decision to add a hiring event to Summer of Nix came at the cost that there was even less time to spend on the main program, requiring to fully rely on and trust that our team structure would work successfully. To lighten it up as much as we could, we reduced the hiring event to the absolute essentials — a no-fuzz moment of connecting one side with the other in the most efficient and direct way. This was born out of necessity, but it probably became a feature, because such an event saved precious time for the participating companies and candidates.

We thus compiled a list of more than 60 companies which, as we got from different sources, had been using Nix. Many of them were small to medium enterprises, but some are also very large, and we reached out to them. Of those 60 companies, 13 agreed to actively participate in the hiring event.

With these companies, spread mostly over North America and Europe, and participants across the globe, another challenge were time zones. We tried to duplicate every event within Summer of Nix to cover all time zones, but for this hiring event, we decided to go for a unique time slot from 17:00 to 20:00 (UTC+2). There is no single slot available that covers West-Coast US to the Japanese time zone, and this one was picked at the expense of Japan, for whom the event unfortunately was very late in the evening.

The hiring event was structured in two big sections: First every company would give a short lightning presentation about themselves, then company representatives would go into on breakout rooms each, their virtual booth, and participants would move through these booths to have direct conversations with them. To make this an interesting event for everyone, we had to make sure that booth audience was evenly spread, so that no company was alone or flooded with the majority of participants at once. That is why we decided to build a schedule for the booth meetings, leaving breaks for flexibility. While the total duration of the event was constrained by time zones, an open variable was the length of the booth meetings. Longer but fewer time slots would mean more participants per meeting, whereas shorter but more time slots could mean a one-on-one interview for everyone. The latter was not possible given the amount of companies and candidates we had, and we thus decided *against* one-on-one meetings, quite happily so because we did not want to create an interview situation, but rather an active and open conversation. But we also wanted to avoid a webinar-style one-directional situation with too many participants at the same time in one booth. We therefore considered optimal between 2 and 5 participants per booth per slot. It turned out that with this setup, the total number of participants in the hiring event, and its total duration, slots needed to be about 15 minutes long, which appeared quick but sufficient to make a first connection.

To build the schedule, we first asked participants and companies for their preferences. The rest then meant filling up a matrix of meeting slots × participants × companies, considering time zones, participant and company preferences at the same time.[^2] With helper scripts and much manual arrangement, we came up with schedules.

[^2] It would be absolutely great to write an online or command line tool for this.

In terms of tooling, after some consideration, we decided against any commercial solution and ran it entirely with the open video conferencing solutions BigBlueButton for the lightning presentations, and Jitsi for the booth breakout rooms.

The event itself was quite a success, and seems to have run smoothly. See the Feedback section for details.

3 Contributions

During the program I and others knew that everyone was doing something*. We got a glimpse of each other's work through daily meetings, chats, and other means of communication. However, a larger perspective of what we have achieved as a group was very difficult to see until after the program. This section tries to provide some insight into this.*

3.1 Code contributions — summary statistics

It would be convenient to simply go through our issue list and check what had been closed, and take that as the program output. For several reasons it was not that easy:

Although the granularity of our issue list was quite homogeneous, one issue corresponded roughly to one NLNet package, there were exceptions such as issues for larger subcomponents or dependencies, and also different interpretations of what "closed" meant. A "90% done" open issue might represent more meaningful work than another closed one, and a finished package with dependencies might stand for one big or multiple smaller closed issues. We also had a "ready-for-review" label as an additional issue state, but since our review team was probably under-staffed, we did not manage to bring all of them all the way to "closed." Still, considering these limitations, the issue list activity *is* an important indicator for Summer of Nix work. Fig. 1 shows this activity, the number of created and closed issues, as a function of time, as well as the number of issues flagged as "ready-for-review" that we ended up with.

Most issues were created (orange line) automatically in two big chunks, mid-2020, a year before starting Summer of Nix, when the original packaging project with independent developers working as contractors started. The issues were created from an internal project database that NLNet maintains of each of the projects they support with grants. The number of issues grew further during the program, because some of these auto-generated issues were broken up into sub-issues, and some projects were added directly by NLNet.

Issues were closed (blue line) for a variety of reasons, primarily because the work had been done, but also for a number of other reasons. For example, some issues were duplicates, others referred to hardware-only projects without

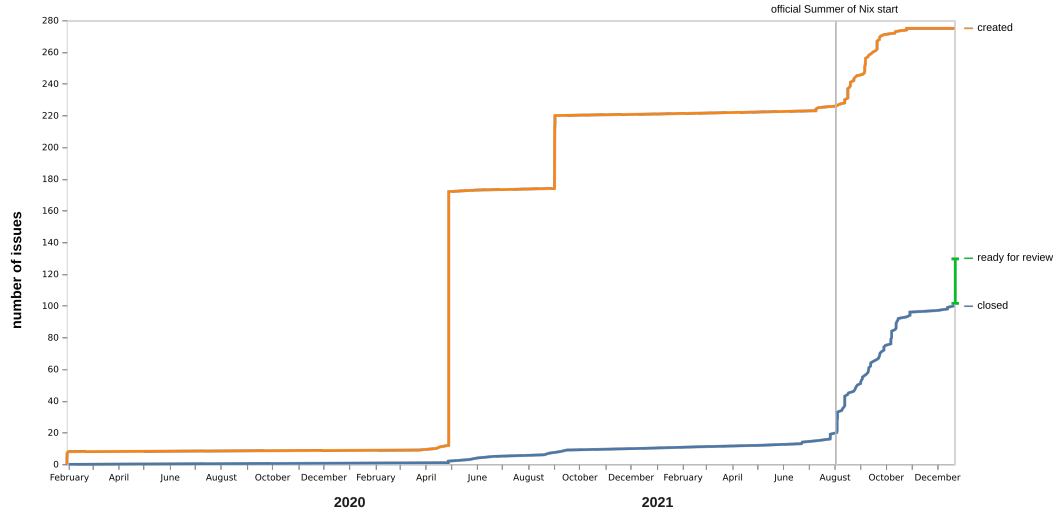

Figure 1: Number of newly created and closed issues in our central issue list as a function of time.

anything to package, others were intractable. Issues that remained in “ready-for-review” state (green marker) at the end of the program are also shown. In absolute numbers, we have started with about 230 open issues, of which ca. 10 were already closed by prior packaging efforts. We ended up with a total number of 276 issues, of which 101 were closed, and 27 flagged as “ready-for-review.”

As discussed above, the number of issues alone does not give a precise image of the work involved, as their size and difficulty varied drastically, even if they were valid packaging requests. Some required simply bumping a version on `nixpkgs` when the associated project had already been packaged there. Others involved profound work, for example bootstrapping, mobile NixOS, work on tooling to tackle insufficiently supported language ecosystems, but also simply small but difficult-to-package programs, and others with many unpackaged dependencies that also had to be delivered. Work on such issues could easily have taken the time of a single packager over the whole program’s time frame and more.

An interesting number, although equally treacherous, is the number of issues closed or flagged as “ready-for-review” *on average per participant*. It was ca. 3.4, and above our expected rate of 3. Again, this number is really only a gross approximation of work, and it does not capture any of the additional non-NGI packages, bug fixes, improved documentation, tooling, and other contributions that came out of it. It also does not account for the quality of the contributions.

Another, similarly limited but still interesting, perspective of the code-contributions done during Summer of Nix is the number of repositories under the `ngi-nix` GitHub organization. Every tool to be reproducibly packaged went under its own repository, although there were exceptions such as upstream `nixpkgs` contributions. The number of repositories went up drastically once the program started, as shown in fig. 2, and many new ones were created over the course of the program. At the time of writing there are in total about 180 repositories, many public but also some private, in the `ngi-nix` organization on GitHub, holding FOSS reproducibly packaged to varying degree. Not all of this work had been finished, and some was adjacent work and not directly an NGI package, which is why this number is higher than the number of issues

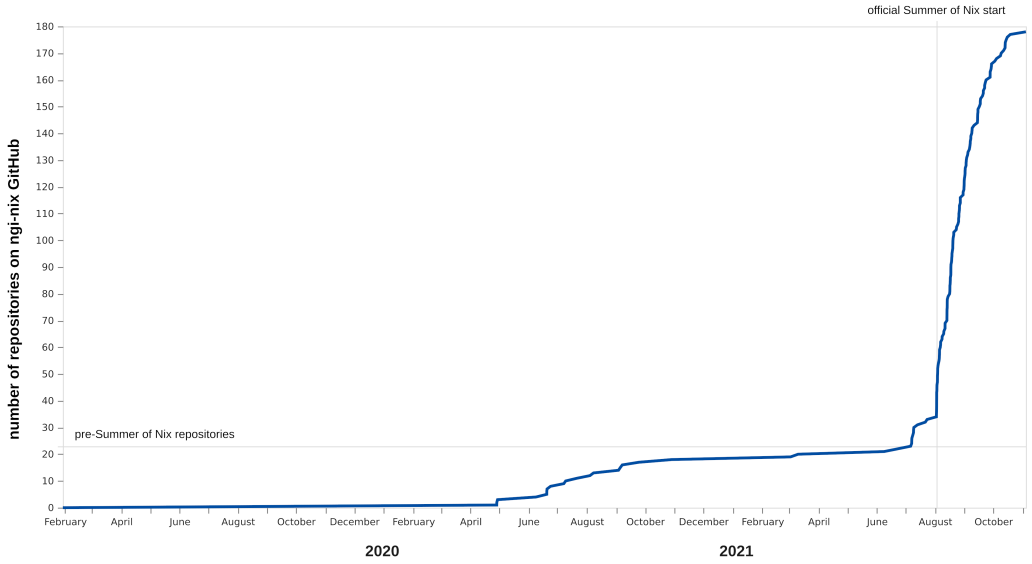

Figure 2: Number of repositories in the ngi-nix GitHub organization as a function of time.

closed or flagged for review.

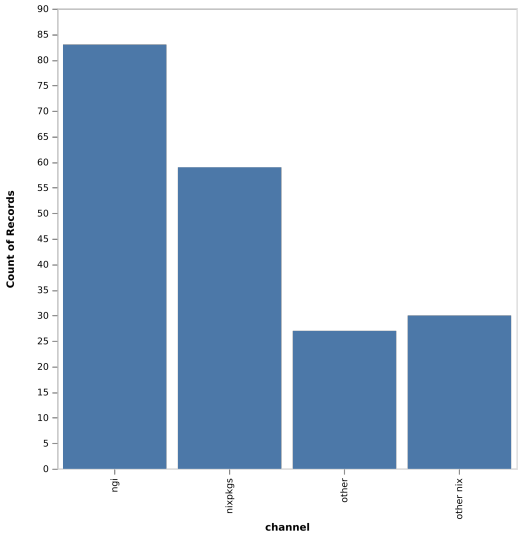

Figure 3: Number of contributions for different upstream channels.

Besides GitHub, another source of information are the time sheets that program participants had filled out during the program. We asked participants to list their contributions there. Although it was tedious, we gathered every reported contribution, an impressive number of 199 repositories, upstream pull requests, or relevant issues to various channels, in structured form from these time sheets. The full list is in the next subsection, sec. 3.2.

Fig. 3 summarizes *where* those contributions went: most are repositories on `ngi-nix`. A lot of contributions, NGI packages, their dependencies, bug fixes or improvements to documentation went directly to `nixpkgs`, Nix' official software library. And finally others were added to the wider Nix ecosystem or directly to upstream repositories.

Most contributions to the Nix community, outside of `nixpkgs`, were bug fixes

on the so-called `lang2nix`[14] helper tools to reproducibly package software from various programming language ecosystems. But also new tools were initiated, like `dream2nix` or static analyzer. Contributions to documentation in hope to simplify future packaging efforts were made, and two RFCs were written with suggestions to evolve the Nix ecosystem.

3.2 Contribution list

3.2.1 Namecoin package

https://www.namecoin.org/

Namecoin is a key/value pair registration and transfer system based on the Bitcoin technology. Bitcoin frees money – Namecoin frees DNS, identities, and other technologies.

- https://github.com/ngi-nix/namecoin-core/pull/1/files

3.2.2 Kazarma

https://gitlab.com/kazarma/kazarma/

A bridge between Matrix and Activity pub.

- https://github.com/ngi-nix/kazarma/pull/1

3.2.3 Lemmy

https://join-lemmy.org/

Lemmy is similar to sites like Reddit, Lobste.rs, or Hacker News. You subscribe to communities you're interested in, post links and discussions, then vote and comment on them.

- https://github.com/ngi-nix/lemmy-ui/tree/flakes
- https://github.com/ngi-nix/lemmy/tree/flakes
- https://github.com/NixOS/nixpkgs/pull/137186
- https://github.com/NixOS/nixpkgs/pull/139174

3.2.4 PeerTube

https://joinpeertube.org/

Free software to take back control of your videos

- https://github.com/NixOS/nixpkgs/pull/119110#issuecomment-961536164
- https://github.com/Chocobozzz/PeerTube/issues/4393
- https://github.com/Chocobozzz/PeerTube/issues/4394

[14] By `lang2nix` the Nix community refers to tools where `lang` is usually substituted with the programming language or its native package manager, such as `poetry2nix` for Python and Poetry, `crate2nix` for Rust and Cargo. These tools automatically convert package descriptions from an ecosystem's package manager to reproducible Nix package descriptions.

3.2.5 Hyperspace

https://www.hyperhyperspace.org/

Make all data local. Communicate only through data sync

- https://github.com/ngi-nix/hyperspace/pull/1
- https://github.com/hypercore-protocol/cli/issues/51

3.2.6 GnuNet

https://www.gnunet.org/en/index.html

A network protocol stack for building secure, distributed, and privacy-preserving applications.

- https://github.com/NixOS/nixpkgs/pull/136365

3.2.7 Interpeer

https://interpeer.io/

A secure and efficient peer-to-peer networking stack that will empower a human centric internet, circumventing the protocols of the past.

- https://github.com/ngi-nix/channeler
- https://github.com/ngi-nix/packeteer
- https://github.com/ngi-nix/liberate

3.2.8 KiwiIRC

https://kiwiirc.com/

Makes Web IRC easy. A hand-crafted IRC client that you can enjoy. Designed to be used easily and freely.

- https://github.com/kiwiirc/kiwiirc/pull/1605

3.2.9 TCN

https://github.com/TCNCoalition/TCN

Specification and reference implementation of the TCN Protocol for decentralized, privacy-preserving contact tracing.

- https://github.com/ngi-nix/TCN

3.2.10 Corteza

https://github.com/cortezaproject/corteza-server

Corteza is the only 100% free, open-source, standardized and enterprise-grade Low-code platform.

- https://github.com/cortezaproject/corteza-server/pull/253

3.2.11 Blink

https://icanblink.com/

A state of the art, easy to use SIP client, a protocol for real-time sessions such as voice, video and messaging appls.

- https://github.com/ngi-nix/blink

3.2.12 Castopod

https://podlibre.org/tag/castopod-host/

An open-source server made for podcasters who want engage and interact with their audience.

- https://github.com/ngi-nix/castopod-host

3.2.13 Jitsi ecosystem

https://jitsi.org/

Completely free video conferencing

- https://github.com/NixOS/nixpkgs/pull/137202
- https://github.com/NixOS/nixpkgs/pull/139948
- https://github.com/jitsi/jibri/issues/441

3.2.14 Liberaforms

https://liberaforms.org/en

Makes it easy to create and manage forms that respect the digital rights of the people who use it.

- https://github.com/ngi-nix/liberaforms
- https://gitlab.com/liberaforms/liberaforms/-/issues/126
- https://gitlab.com/liberaforms/liberaforms/-/merge_requests/221

3.2.15 Etesync

https://www.etesync.com/

Secure, end-to-end encrypted, and privacy respecting sync for your contacts, calendars, tasks and notes.

- https://github.com/NixOS/nixpkgs/pull/133536
- https://github.com/ngi-nix/libetebase
- https://github.com/ngi-nix/etebase-rs
- https://github.com/ngi-nix/libetebase
- https://github.com/ngi-nix/etebase-java
- https://github.com/NixOS/nixpkgs/pull/134343

3.2.16 liboqs

https://openquantumsafe.org/

An open source C library for quantum-safe cryptographic algorithms

- https://github.com/ngi-nix/liboqs

3.2.17 Milagro

https://github.com/apache/incubator-milagro

A set of crypto libraries and core security infrastructure applications, built for decentralized networks yet suitable for enterprises.

- https://github.com/ngi-nix/incubator-milagro

3.2.18 Software Heritage

https://www.softwareheritage.org/

-
- https://github.com/ngi-nix/swh-environment/pull/1

3.2.19 DiffPrivacyInference

https://github.com/DiffMu/DiffPrivacyInference.jl

A type checker which can automatically analyze Julia programs with respect to differential privacy guarantees.

- https://github.com/ngi-nix/DiffPrivacyInference/

3.2.20 Kaidan

https://www.kaidan.im/

A user-friendly and modern chat app for every device. It uses the open communication protocol XMPP (Jabber).

- https://github.com/NixOS/nixpkgs/pull/136916

3.2.21 Weblate

https://weblate.org/en/

A web based translation tool, used by over 2,500 libre software projects and companies in over 165 countries.

- https://github.com/ngi-nix/weblate
- https://github.com/WeblateOrg/weblate/issues/6679

3.2.22 LibreSOC

https://libre-soc.org/

With Libre-SOC, you can take complex algorithms usually intended for power hungry servers with big fat GPUs, and run them on tiny devices like smart-watches, cellphones, and pocket drones without changing your code at all.

- https://github.com/ngi-nix/libresoc-soc/pull/1
- https://git.libre-soc.org/?p=soc.git;a=commit;h=506c93ac12aa4d52db5ba

3.2.23 Coriolis

http://coriolis.lip6.fr/

A complete toolchain for vlsi design. It provides a vhdl compiler and simulator, logic synthetiser, automatic place & route and portable cmos library.

- https://gitlab.lip6.fr/vlsi-eda/coriolis/-/merge_requests/8

3.2.24 Meta Press

https://www.meta-press.es/

Decentralized search engine & automatized press reviews

- https://github.com/ngi-nix/meta-press

3.2.25 OpenFoodFacts

https://world.openfoodfacts.org/

A food products database made by everyone, for everyone. You can use it to make better food choices, and as it is open data, anyone can re-use it for any purpose.

- https://github.com/ngi-nix/openfoodfacts-server/tree/ngi-nix

3.2.26 Nyxt

https://nyxt.atlas.engineer/

Power-browser that ships with tens of features that allow you to quickly analyze, navigate, and extract information from the Internet.

- https://github.com/ngi-nix/nyxt

3.2.27 ForgeFed

https://forgefed.peers.community/

An upcoming federation protocol for enabling interoperability between version control services.

- https://github.com/ngi-nix/mcfi
- https://github.com/ngi-nix/forgefed

3.2.28 CVEHound

https://github.com/evdenis/cvehound

A tool for checking linux sources for known CVEs (Common Vulnerabilities and Exposures).

- https://github.com/ngi-nix/cvehound/

3.2.29 trackingthetrackers

https://nlnet.nl/project/F-Droid-Trackers/

A free software, community app store on Android that has been working since 2010 to make all forms of tracking and advertising visible to users.

- https://github.com/ngi-nix/trackingthetrackers-apk-api
- https://github.com/ngi-nix/trackingthetrackers-scripts
- https://github.com/ngi-nix/trackingthetrackers-etl
- https://github.com/ngi-nix/trackingthetrackers-classifier
- https://github.com/ngi-nix/trackingthetrackers-extracted-features

3.2.30 movim

https://nlnet.nl/project/Movim-OMEMO/

A web platform that delivers social and IM features on top of the mature XMPP standard (aka Jabber)

- https://github.com/ngi-nix/movim

3.2.31 url-frontier

https://github.com/crawler-commons/url-frontier

A web crawler-neutral API

- https://github.com/ngi-nix/url-frontier

3.2.32 Katzenpost

https://katzenpost.mixnetworks.org/

Mix network protocol libraries. What is a mix network? It is an anonymous communications system.

- https://github.com/ngi-nix/katzenpost

3.2.33 Variation Graph

https://github.com/vgteam/vg

Tools for working with genome variation graphs

- https://github.com/ngi-nix/magic_rb-vg

3.2.34 Xemu for MEGA65 Phone

https://github.com/lgblgblgb/xemu

Emulations of mainly 8 bit machines, including the Commodore LCD, Commodore 65, and the MEGA65 as well.

- https://github.com/ngi-nix/xemu

3.2.35 rebuilderd

https://rebuilderd.com/

Independent verification of binary packages - reproducible builds

- https://github.com/ngi-nix/rebuilderd

3.2.36 ricochet

https://github.com/blueprint-freespeech/ricochet-refresh

Anonymous peer-to-peer instant messaging

- https://github.com/ngi-nix/ricochet

3.2.37 poliscoops

https://github.com/openstate/poliscoops

An interactive online platform that allows journalists and citizens to stay informed, and keep up to date with the growing group of political parties and politicians relevant to them.

- https://github.com/ngi-nix/poliscoops-flake

3.2.38 GoatCounter

https://www.goatcounter.com/

Easy web analytics. No tracking of personal data.

- https://github.com/ngi-nix/magic_rb-goatcounter

3.2.39 Simple Bandwidth Scanner (OnBaSca)

https://tpo.pages.torproject.net/network-health/sbws/README.html

Tor bandwidth scanner

- https://github.com/ngi-nix/sbws-flake

3.2.40 NetFilter

https://www.netfilter.org/

Provides packet filtering software for the Linux 2.4.x and later kernel series. The netfilter project is commonly associated with iptables and its successor nftables.

- https://github.com/NixOS/nixpkgs/pull/140994

3.2.41 Bitcoin Transactions

https://peersm.com/wallet

Javascript implementation of the Bitcoin protocol for any Bitcoin based coins, on server and inside browsers

- https://github.com/ngi-nix/bitcoin-transactions

3.2.42 Anastasis

https://anastasis.lu/en/

A key recovery system that allows the user to securely deposit shares of a core secret with an open set of escrow providers, to recover it if the secret is lost

- https://github.com/ngi-nix/anastasis
- https://github.com/ngi-nix/anastasis-gtk

3.2.43 GNU Taler

https://taler.net/en/

A payment system that makes privacy-friendly online transactions fast and easy

- https://github.com/NixOS/nixpkgs/pull/140477

3.2.44 Arpa2

http://arpa2.net/

Tools to repopulate a decentralised global internet that offers security and privacy by design.

- https://github.com/ngi-nix/arpa2
- https://github.com/NixOS/nixpkgs/pull/134965
- https://github.com/NixOS/nixpkgs/pull/134332

3.2.45 P2Pcollab

https://p2pcollab.net/

Protocols for peer-to-peer collaboration.

- https://github.com/ngi-nix/p2pcollab
- https://github.com/p2pcollab/ocaml-noise-socket/issues/4
- https://github.com/p2pcollab/ocaml-blip/pull/1
- https://github.com/p2pcollab/ocaml-urps/pull/2
- https://github.com/p2pcollab/ocaml-fsq/issues/2
- https://github.com/p2pcollab/ocaml-sunnyhash/pull/2
- https://github.com/ngi-nix/bloomf/tree/flake-only

3.2.46 CWE checker

https://github.com/fkie-cad/cwe_checker

A security checker written in Rust using Ghidra

- https://github.com/ngi-nix/cwe_checker/tree/flake-only
- https://github.com/fkie-cad/cwe_checker/pull/234

3.2.47 Manyverse

https://www.manyver.se/

A social network off the grid.

- https://github.com/ngi-nix/manyverse/tree/nix/napalm-mazurel-2

3.2.48 EEZ ecosystem

https://www.envox.eu/

The modular BB3 (Bench Box 3) test & measurement solution which in combination with the cross-platform EEZ Studio offers an appealing open source feature packed framework for automating everyday testing and development tasks suitable for makers, hobbyists, students and professionals.

- https://github.com/ngi-nix/studio/tree/flake-only

3.2.49 DNSSEC

https://github.com/NLnetLabs/dnssec-ceremony-tools/

DNSSEC provides trust in the DNS by guaranteeing the authenticity and integrity of DNS responses.

- https://github.com/ngi-nix/dnssec-ceremony-tools/tree/flake

3.2.50 URLExtract

https://github.com/lipoja/URLExtract

URLExtract is python class for collecting (extracting) URLs from given text based on locating TLD.

- https://github.com/NixOS/nixpkgs/pull/139814

3.2.51 Owncast

https://owncast.online/

A self-hosted live video and web chat server for use with existing popular broadcasting software.

- https://github.com/ngi-nix/owncast/tree/cleanup
- https://github.com/ngi-nix/owncast-admin/tree/flake

3.2.52 BigBlueButton

https://bigbluebutton.org/

A global teaching platform. It was developed in a school, not in a boardroom. Making it the only virtual classroom built from the ground up, just for teachers.

- https://github.com/ngi-nix/bbb4nix/tree/flake

3.2.53 Briar

https://briarproject.org/

A secure messaging app designed for activists, journalists and civil society groups. Instead of using a central server, encrypted messages are synchronized directly between the users' devices, protecting users and their relationships from surveillance.

- https://github.com/ngi-nix/briar

3.2.54 elRepo.io

https://elrepo.io/

Resilient, human-centered, distributed content sharing and discovery.

- https://github.com/ngi-nix/elRepo.io

3.2.55 IMSI Pseudonymization

https://osmocom.org/

Open Source Mobile Communications

- https://github.com/ngi-nix/imsi-pseudo

3.2.56 Conversations

https://conversations.im/

An Android client for the federated, provider independent network of instant messaging servers that use the Extensible messaging and Presence Protocol (XMPP).

- https://github.com/ngi-nix/Conversations-1/tree/flake-only

3.2.57 Reowulft

https://reowolf.net/

Replace a decades-old application programming interface (BSD-style sockets) for communication on the Internet. In this project, a novel programming interface is implemented at the systems level that is interoperable with existing Internet applications.

- https://github.com/ngi-nix/reowolf

3.2.58 Mailpile

https://www.mailpile.is/

Mailpile is an e-mail client! Mailpile is a search engine and a personal webmail server. Mailpile is an easy way to encrypt your e-mail. Mailpile is software you run yourself, on your own computer.

- https://github.com/NixOS/nixpkgs/pull/135543

3.2.59 mobile-nixos

https://github.com/NixOS/mobile-nixos

Mobile NixOS is a superset on top of NixOS Linux, Nixpkgs and Nix, aiming to abstract away the differences between mobile devices. In four words “NixOS, on your phone.”

- https://github.com/NixOS/mobile-nixos/pull/404

3.2.60 OSX-KVM

https://github.com/foxlet/macOS-Simple-KVM

Tools to set up a quick macOS VM in QEMU, accelerated by KVM.

- https://github.com/ngi-nix/OSX-KVM

3.2.61 Waasabi

https://waasabi.org/

Open source framework for custom live streaming events and conferences.

- https://github.com/ngi-nix/waasabi

3.2.62 Wireguard

https://www.wireguard.com/

An extremely simple yet fast and modern VPN that utilizes state-of-the-art cryptography. It aims to be faster, simpler, leaner, and more useful than IPsec, while avoiding the massive headache.

- https://github.com/NixOS/nixpkgs/pull/133763

3.2.63 SearX

https://searx.me/

Privacy-respecting metasearch engine

- https://github.com/ngi-nix/searx-thegreenopenweb
- https://github.com/ngi-nix/tgwf-searx-plugins/tree/nix-flake

3.2.64 PCB-rnd

http://repo.hu/projects/pcb-rnd/

Is a free/open source, flexible, modular Printed Circuit Board editor.

- https://github.com/ngi-nix/pcb-rnd/pull/2

3.2.65 Noise Explorer

https://noiseexplorer.com/

An online engine for reasoning about Noise Protocol Framework (revision 34) Handshake Patterns.

- https://github.com/ngi-nix/noiseExplorer

3.2.66 Universal Resolver

https://danubetech.com/

Resolve any DID on any blockchain, DLT, or other decentralized network.

- https://github.com/ngi-nix/universal-resolver

3.2.67 IRMA

https://irma.app/

Everyone can freely download the IRMA app (Android and iOS and F-droid) and fill it with their own data. For instance, store your email address and possibly mobile number as attribute in your IRMA app.

- https://github.com/ngi-nix/irmago-server
- https://github.com/ngi-nix/IRMAseal

3.2.68 Delta Chat

https://delta.chat/en/

Delta Chat is like Telegram or Whatsapp but without the tracking or central control. Delta Chat does not need your phone number. Check out our privacy statement.

- https://github.com/ngi-nix/deltabot

3.2.69 IPFS and IPFS Search

https://ipfs-search.com/#/

A Free and Open Source (FOSS) search engine for directories, documents, videos, music on the Interplanetary Filesystem (IPFS)

- https://github.com/ngi-nix/ipfs-search
- https://github.com/NixOS/nixpkgs/pull/135850
- https://github.com/NixOS/nixpkgs/pull/135867
- https://github.com/NixOS/nixpkgs/pull/136137
- https://github.com/NixOS/nixpkgs/pull/136143
- https://github.com/NixOS/nixpkgs/pull/136170
- https://github.com/NixOS/nixpkgs/pull/136255
- https://github.com/NixOS/nixpkgs/pull/136261
- https://github.com/NixOS/nixpkgs/pull/137070

3.2.70 Meilisearch

https://www.meilisearch.com/

Powerful, fast, and an easy to use search engine

- https://github.com/NixOS/nixpkgs/pull/136353
- https://github.com/NixOS/nixpkgs/pull/137612
- https://github.com/NixOS/nixpkgs/pull/137078
- https://github.com/NixOS/nixpkgs/pull/138778

3.2.71 Cryptpad

https://cryptpad.fr/

End-to-end encrypted and open-source collaborative documents.

- https://github.com/NixOS/nixpkgs/pull/133202
- https://github.com/NixOS/nixpkgs/issues/128154

3.2.72 Sonar

https://arso.xyz/

We’re developing tools to make it easy to share media collaboratively without relying on centralized services.

- https://github.com/ngi-nix/sonar-1
- https://github.com/ngi-nix/sonar-tantivy/pull/1

3.2.73 Webshell

https://wiki.ljudmila.org/WebShell

A free, open-source, private and secure alternative to commercial cloud storage and web-based software. It's a web desktop which can be self-hosted on your server to provide access to your files and apps from the web.

- https://github.com/ngi-nix/webshell

3.2.74 Misskey

https://join.misskey.page/en-US/

A decentralized and open source microblogging platform

- https://github.com/ngi-nix/misskey/tree/flake

3.2.75 EgilSCIM

https://github.com/Sambruk/EgilSCIM

The EGIL SCIM client implements the EGIL profile of the SS 12000 standard. It reads information about students, groups etc. from LDAP and sends updates to a SCIM server.

- https://github.com/Sambruk/EgilSCIM/pull/118
- https://github.com/Sambruk/EgilSCIM/pull/121
- https://github.com/Sambruk/EgilSCIM/pull/122
- https://github.com/Sambruk/EgilSCIM/pull/123
- https://github.com/ngi-nix/EgilSCIM/tree/flake-only

3.2.76 Other flakes on the ngi organization

Mostly dependencies of NGI projects

- https://github.com/ngi-nix/pymilter
- https://github.com/ngi-nix/pgp-milter
- https://github.com/ngi-nix/libspng
- https://github.com/ngi-nix/tika

3.2.77 2nix tooling

Contributions to various 2nix tools.

- https://github.com/svanderburg/composer2nix/pull/24
- https://github.com/svanderburg/composer2nix/issues/21#issuecomment-93
- https://github.com/nix-community/napalm/pull/35
- https://github.com/svanderburg/node2nix/pull/261
- https://github.com/nix-community/npmlock2nix/pull/93
- https://github.com/nix-community/npmlock2nix/pull/94
- https://github.com/nix-community/npmlock2nix/pull/108,
- https://github.com/nix-community/npmlock2nix/pull/110,
- https://github.com/nix-community/npmlock2nix/pull/111,

- https://github.com/nix-community/npmlock2nix/pull/115
- https://github.com/nix-community/poetry2nix/pull/368
- https://github.com/fzakaria/mvn2nix/issues/40
- https://github.com/nix-community/naersk/issues/187
- https://github.com/nix-community/naersk/issues/188
- https://github.com/rvl/bower2nix/pull/21
- https://github.com/nix-community/dream2nix

3.2.78 Flake templates

Nix flake templates

- https://github.com/ngi-nix/flakes-templates/pull/1
- https://github.com/NixOS/templates/pull/7
- https://github.com/ngi-nix/nixos-modules-flake-template/blob/main/fla

3.2.79 Other nixpkgs contributions

https://github.com/NixOS/nixpkgs

Docs, broken, version bumps, adding dependencies

- https://github.com/NixOS/nixpkgs/pull/126758
- https://github.com/NixOS/nixpkgs/pull/127724
- https://github.com/NixOS/nixpkgs/pull/130419
- https://github.com/NixOS/nixpkgs/pull/132287#pullrequestreview-736233
- https://github.com/NixOS/nixpkgs/pull/132906
- https://github.com/NixOS/nixpkgs/pull/133354
- https://github.com/NixOS/nixpkgs/pull/133593
- https://github.com/NixOS/nixpkgs/pull/136274
- https://github.com/NixOS/nixpkgs/pull/133753
- https://github.com/NixOS/nixpkgs/pull/134110
- https://github.com/NixOS/nixpkgs/pull/134384
- https://github.com/NixOS/nixpkgs/pull/134657
- https://github.com/NixOS/nixpkgs/pull/135573
- https://github.com/NixOS/nixpkgs/pull/135730#pullrequestreview-739660
- https://github.com/NixOS/nixpkgs/pull/136476
- https://github.com/NixOS/nixpkgs/pull/136524
- https://github.com/NixOS/nixpkgs/pull/136632
- https://github.com/NixOS/nixpkgs/pull/136637
- https://github.com/NixOS/nixpkgs/pull/136703
- https://github.com/NixOS/nixpkgs/pull/138016
- https://github.com/NixOS/nixpkgs/pull/138587
- https://github.com/NixOS/nixpkgs/pull/138591
- https://github.com/NixOS/nixpkgs/pull/138707
- https://github.com/NixOS/nixpkgs/pull/138750
- https://github.com/NixOS/nixpkgs/pull/140208
- https://github.com/NixOS/nixpkgs/pull/141822
- https://github.com/NixOS/nixpkgs/pull/142393
- https://github.com/NixOS/nixpkgs/pull/144547
- https://github.com/NixOS/nixpkgs/pull/144314

3.2.80 Other Nix contributions

-
- https://github.com/NixOS/nix/pull/5364
- https://github.com/NixOS/nixos-search/pull/360
- https://github.com/nprindle/nix-parsec/pull/4
- https://github.com/NixOS/rfcs/pull/99
- https://github.com/NixOS/rfcs/pull/100
- https://github.com/nix-community/home-manager/issues/2251
- https://github.com/seppeljordan/nix-prefetch-github/issues/40
- https://github.com/NixOS/nix/issues/5169
- https://github.com/NixOS/nix/pull/5286
- https://github.com/NixOS/nix/pull/5163#issuecomment-924488262
- https://github.com/Mic92/nixos-shell/pull/41

3.2.81 Other

Other upstream contributions

- https://github.com/bytefury/crater/issues/578
- https://github.com/bytefury/crater/pull/579
- https://github.com/YosysHQ/prjtrellis/pull/178
- https://github.com/unbit/uwsgi-docs/pull/499
- https://github.com/celery/celery/pull/6988
- https://github.com/electron/electron/issues/31121
- https://github.com/Mazurel/android2nix

3.3 Non-code contributions

In addition to just participating, one participant ran a Nix camp where one could hack on-site together. Another wrote a blog post for the Summer of Nix website. Yet another participant helped to record all video sessions, which turned out to be great learning material, although unfortunately not intended for public use.

3.4 Outcome for the participants

If it is hard to evaluate the code-contributions of the program, it is even more difficult to say what the outcome was for individual participants. Some greatly enjoyed the program, became first-time maintainers on `nixpkgs`. Others got hired either by the teams of the software that they packaged during Summer of Nix, or through the hiring event. Some did not enjoy it as much and probably did not take out a lot. We do not have exact numbers, not even a vague idea.

Directly from the time sheets, we could see that in total four participant dropped out — two of them before the program started. Not everyone finished the program fully, either. The participant contract was designed to deliver 320 hours in two chunks of 160 hours. Almost everyone delivered the full chunk of the first half, but 5 participants finished the second part only partially for various reasons. This means that we initially had 31 participants working, and fewer at the end of the program.

4 Feedback

We organized four open feedback sessions after Summer of Nix to gather thoughts about the program from the perspective of participants and mentors — ultimately the core groups of the program.

The overall experience was diverse, mostly positive, but also different from person to person and team to team. Some wider aspects of the program were universally seen as positive, and their absence regretted where their implementation in practise did not work out:

- Participants universally appreciated in particular the **team experience, active communication, and not working alone**. Some teams fared much better here than others, providing lively discussions, a weekly rhythm, and a good support structure. But even in these teams, moments when participants, in particular those with less Nix experience, had to work alone on a package were seen as difficult. Other teams were much quieter and therefore the participants had a harder time. Similarly, from the mentor perspective a quieter team with star-shaped communication instead of a horizontal support structure meant more work for the mentor because they were the principal counterpart for all team members. The same theme, either appreciation of good support communication or the lack of it, came up in various aspects: Some participants felt lost and, on the other hand, mentors said that they had trouble of getting participants to ask for help early when they were blocked. It is thus not surprising that all participants appreciated pair programming sessions, which we later-on in the programming organized via two random, cross-team pairings per week, and lamented their lack in the beginning of the program. To summarize, it seems team work sometimes went very well, sometimes not, and while it did not go badly, there is certainly room for improvement.
- The centrally organized presentations about the fundamentals of Nix by core developers, with an interactive Q&A session attached, were mentioned as highlights of the program. This was certainly not only due to the quality of content, but also due to the fact that this was a moment of community and also something that, similar to regular team meetings, gave a certain rhythm to the program. **Rhythm and structure** was a central aspect of the feedback in general. It seems that more rhythm and structure would have been beneficial. Presentations, team meetings, and then later-on cross-team pairing sessions were all appreciated. In addition it was mentioned that motivation tapered off towards the end of the program, which might have been less pronounced if there were events like a closing ceremony to work towards. We discussed but did not conduct final presentations. A mentor mentioned that the first week was very busy, so that they were barely able to work on their day job, and after a while the team got bored and work tapered off.
- Work distribution was another central aspect that generated some frustration: although the general assign-and-trade approach worked fine, some larger package families, that is, packages from a similar programming language or software ecosystem, were spread across teams. It took a while to all bring them to the same people to tackle them jointly. And with the GitHub issue list alone it was difficult to see who worked on what and when. Furthermore, some participants would have liked to specialize on packaging software written in specific programming languages and we had no notion of that in our work distribution. On the other hand, we had the impression that some also enjoyed peeking into diverse and new technology. Probably a common denominator was that there was a perceived lack

of ecosystem-oriented communication beyond team boundaries, such as a simple Matrix channel to discuss JavaScript packaging, so that every one has **an opportunity to follow their tech interests**.
- Another aspect related to motivation were the actual packages we worked on. Some of them, such as Jitsi, BigBlueButton, Arpa2 are very big, well-known tools, but sometimes demotivating because they are difficult to package due to their size and complexity. Inversely, others were small, prototype-like tools without regular maintenance, where the question came up why we were spending time on them. Many felt that it would be worth investing more time in central projects for the Nix community that simplify packaging in general. This became particularly clear when similar questions came up over and over from different participants. It was frustrating not to work on improving those directly. We somewhat allowed this, if it directly or indirectly helped for NGI packages and if someone had already worked on many "official" packages. But we did not encourage it either, because we were bound by the goals of the funding grant. **A clear answer to why we are working on something** would have been appreciated.
- Some participants spent a significant amount of time on non-Nix and non-code tasks, querying upstream maintainers and such. Few actually liked this part of the work and would have appreciated contacting project upstream maintainers beforehand to get cleaner task descriptions, or support with communication skills and what someone called "software archaeology." Some packages were also just not possible to package, e.g. due to being hardware related, and it was puzzling for some to be assigned to do something which is not possible. **Clearer task descriptions** would have been appropriate.
- we had no formal roles in Summer of Nix, besides participants, mentors and administration. However, some participants did take initiative and responsibility to organize things such as recording all presentations and distributing them, hosting code and chat sessions, and more. Some were sliding into these roles, and would have appreciated a more structured role distribution to know clearly what their tasks would be.
- Some aspects of the packaging workflow and work organization were not ideal. For example, the choice between external and internal Nix flakes, and between separate repositories and a big monorepo.

Then a few points about Nix came up:

- The Nix documentation was universally seen as difficult to read on many aspects.
- Mediocre support for various ecosystems and bugs, and lack of documentation, templates, and tutorials on how to use the various `lang2nix` tools. Simply finding and picking the right tool was a challenge as well because there is no unified documentation or recommendation, and there exists many tools with similar purpose and naming. Specific ecosystems that were mentioned are: Racket, Java, Android, and JavaScript.
- Nix flakes restrictions were not a problem because it was part of the learning experience

Notes and suggestions from mentors:

- Expect lots of similar questions: how to do overlays, how to do `X`. Also on ecosystems and tooling. Explain common themes such as `lang2nix`, services, nixos-container, modules, caches/Hydra to participants.

- Get people to communicate, ideally in GitHub issues, but some feel it it very permanent and thus do not use it. Matrix is less overhead, easier, less formal and might be necessary to jump-start working.
- Some are afraid to ask for help. Foster a culture of open exchange, organize a regular checkin with report on yesterday, plan for today, blockers.
- Explain programming vs. packaging skills: communication and software archaeology.
- Show and help how to "sell" Nix adoption to upstream

Concrete ideas for a future edition:

- The introduction system (having a suggestion of two people to meet per week) was very successful. If possible it should start at the beginning. This is one way that you can find other people that might help you when you get stuck.
- If some people become long time members of the Nix community that is a success.
- Clearer rules about the general and the off-topic channels in the chat. Many participants were in there and the rooms were flooded by messages.
- External communication in the form of blog posts would be great and also motivating. We only published a single article and did not encourage this much.
- Assign multiple people per package. It was amazing to collaborate with others on packages.
- Get common compute resources, e.g. from a cloud provider, and actually have the service running on servers maintained by program organizers or participants would be great .
- Facilitator for pairing meetings would be great because meetings were perceived as very beneficial.
- Move to external Nix flakes and a monorepo, or a metaflake .
- Have an official internal documentation/timesheet page per participant
- Have both, teams (time zones) and cross-team communities of interest (language/ecosystem)
- Better upfront issue and team assignment
- Use automatic ryantm-style update PRs or CI
  - e.g.: https://github.com/DeterminateSystems/update-flake-lock
- Make use of the content or experiences that we make during the program. For example, publish the presentations and produce quality content for the community.
- Staggered beginning to handle the initial workload
- Build better onboarding material, similar to the Rustlang book, but for Nix
- Permanent Jitsi channel for spontaneous encounters
- Every participant could give half an hour to explain sometime for other participants on any topic so that they have to learn about it and that there is more exchange cross-teams
- Include work on upstream documentation as part of project tasks
- Part of Summer of Nix could be about getting commit permissions for `nixpkgs`
- Summer of Nix could prepare participants for a NixCon conference talk
- Work outside of NGI would be interesting. Summer of Nix should be more than NGI. Companies could submit projects to this event. Call for projects?
- Organizing marketing with an aesthetic dimension could also be interesting. Community manager could organize some polls or games. If too many diverse people are working on documentation, writing blog posts etc., new users can be lost quickly due to different styles and so on.

- Have a follow-up period with a second stipend to *write*. Provide professional support with that.
- One "tech mentor" per programming language. The mentors were really helpful and knowledgeable, but we felt that our problem were very programming language specific (e.g. "I'm getting this weird error with some JavaScript package, what do I do?"). Having a mentor of reference with experience in a particular area of the ecosystem would be quite helpful. Potentially there could be a Java mentor, a JavaScript mentor, a C mentor… One of the most frustrating points was getting stuck, and having somebody who experienced particular problems would have been very helpful. Note that we think the mentors did a great job, this could just be the cherry on the cake.
- Take interests into account when making teams. Some people have an affinity/experience for some a programming language, trying to make teams that are interested in the same area (i.e. the Rust team) could lead to synergies (people sharing useful information they found about a certain thing) and more resilience to frustration ("I like Rust so I will work harder to make this work"). Of course some teams might be more crowded than others, but that could be another criteria to select participants (e.g. "We don't have space in the Rust team unfortunately, but would you be interested in joining the JavaScript team?"). Note that you could have a team interested in Rust, but within the team some people are experienced with Python, and so you could give them a Python and Rust package. It will also improve the willingness to maintain a package in the long-term in case someone is interested in the technologies. The teams were formed by geography — we are not sure it' is important, as long as time zones overlap by 4 hours during the day, and it should be enough to enable all the required meetings.
- Another, more basic nixos-module presentation. The nixos-module presentation was truly on another level. However it dealt with abstract information around the nixos-modules. There should be a "boring" presentation that goes through some existing modules and explains what is happening (e.g."`systemd` has a `requires` attribute that does…","In this module you can see how to initialise a database"). All this information can be of course found in `nixpkgs`, but a detail of the most common "tricks" might go a long way.
- Pre-screening the packages for unmaintained ones. It was confusing to see unmaintained packages in issues. Pre-screening for only the relevant packages might improve the motivation of participants. Getting stuck on a maintained package is one thing. Getting stuck on something that feels it will never be used anyway reduces motivation significantly.
- Organise a pre-packaging effort to help upstream projects to be easier to package.
- New participant role "community manager": someone who animates the Summer of Nix community, organizing discussions, polls, games.
- New participant role "pairing partner": experienced Nixer available for pairing.
- New role "web master|: someone who administers an open website on a repository where everyone can contribute. Move website to static site generator.

5 Budget

We delivered Summer of Nix slightly under budget, because of a few participants leaving prematurely and some who decided to volunteer and leave the money in the original pool. The anticipated budget for all of this work was 131,755€. The final expenses amounted to 107,700€, including mentor and participant stipends, as well as extra costs for material and organization.

6 Conclusions

Overall, we are quite happy with how the program turned out. We knew that we had a tight time budget for organization and also limited experience, but we also knew that it was a unique opportunity to have a pre-existing budget and a group of people interested in doing an event. Under these constraints — and we hope this report makes it clear that organization is "the art of the possible" *within constraints* — after gathering the data, we are amazed about what we have accomplished:

First of all, despite highs and lows, it appears to have been an overall fun and enriching event for participants. Much of the feedback on possible improvements can be traced back to a lack of time on the organizational side, the fact that we were designing this program from scratch, and that everyone was new in their respective roles. We expected that this would be the case the first time. There was little concern about the general idea and set up of the program, and the principal goals of the program. **Learn, meet, deliver** seemed to be appreciated, as well as working with FOSS software, and also the implementation wherever it worked well.

Then, independent of the work output, a measure of success for the Summer of Nix for us was whether we can bring new long term members to the community — maybe even professionally, and certainly we succeeded with some in this regard.

In our view, a future edition should therefore focus on improving the implementation and organization of the program to make most out of everyone's valuable time. Judging from the feedback sessions and personal conversations, there is good consensus on what could be improved. The main bottleneck to actually do so are the organizational capacities that we have to realize these ideas under budget and other constraints. Distributed decision making will be central to achieve this. However, one central constraint that we were all newcomers to designing a program is now overcome, and should immediately improve the situation — although a second round may incur a drop in motivation.

Personally, I am really grateful to have been given the chance and trust to organize this program by my employer Tweag, the NixOS and NLNet Foundations, and ultimately the European Commission, but especially by the participants, mentors, and volunteers, who actually ran this program and in the end achieved an amazing amount of contributions.

We hope that together we were able to make a substantial and lasting contribution to the Free and Open Source Software ecosystem, and that we can continue to doing so in a variety of forms.

References

"Nix: A Safe and Policy-Free System for Software Deployment." n.d.

European Commission. "The European Commission's Open Source Strategy for 2020-2023," n.d. https://ec.europa.eu/info/sites/default/files/en_ec_open_source_strategy_2020-2023.pdf.

———. "What the European Commission Does," n.d. https://ec.europa.eu/info/about-european-commission/what-european-commission-does_en.

Google. "Google Summer of Code," n.d. https://summerofcode.withgoogle.com/.

Nixos Community. "NixOS Website," n.d. https://nixos.org/guides/how-nix-works.html.

NixOS Foundation. "Mission Statement," n.d. https://github.com/NixOS/nixos-foundation.

NLNet. "Foundation Website," n.d. https://nlnet.nl/.

Roberto Di Cosmo et al. "The Mancoosi Project," n.d. https://www.mancoosi.org/.

The Next Generation Internet website. "About Ngi0," n.d. https://www.ngi.eu/ngi-projects/ngi-zero/.

———. "About the NGI Initiative," n.d. https://www.ngi.eu/about/.

The NixOS discourse. "The Summer of Nix—Learn Nix While Doing Useful, Paid Work," n.d. https://discourse.nixos.org/t/the-summer-of-nix-learn-nix-while-doing-useful-paid-work/12225.

The NLNet website. "About PET," n.d. https://nlnet.nl/PET/.

Tweag. "Company Website," n.d. https://tweag.io.

YouTube. "Linus Torvalds on Why Desktop Linux Sucks," n.d. https://youtu.be/Pzl1B7nB9Kc.