Einstieg ins Open-Source-Universum

Einer Community beitreten, GitHub/GitLab etc., Ressourcen über Open-Source-Software, Tools und Literatur

- Introduction to Git [EN]
- Linksammlung Einführung in Git
- In fünf Schritten zur barrierefreien Software
- How to work with a mentor – Interview with Common Syllabi

□□ Introduction to Git [EN]

Git is a software used by developers to work together on a project and keep track of the most recent version of their projects. Git is often seen in combination with Github. Github is a website that adds some commands and features to Git. It also makes the use and interaction with Git more intuitive and visually nicer. Git can however also be used without using Github or any other hosted service at all.

The aim of Git is to keep track of code changes, especially when more than one developer works on the same project. Git works through **repositories**, namely folders, which are either local or remote. Developers work on their own local repositories and then share the last version of their work **pushing** their work on the remote repository through a **commit.** Commits are the way Git keeps track of changes. Every commit is like a checkpoint: at every commit Git saves the new version of the code, creating a code history. Through the history it is possible to access any previous version of the code. The remote repository contains the most updated version of the code and all developers can save it on their local machine by **pulling** it. Git takes care of merging all versions of the code avoiding conflicts.

A powerful feature of Git is the possibility of opening different branches. New branches are used so that the work on the main branch remains unaffected when making new tries. For example, if a developer wants to introduce a new feature but does not know yet if the code will work, they can open a new branch and work on it until the code works. It is possible to merge the branches in a later stage.

Git is very useful for developers but it can be somehow not intuitive to use, at least at the beginning. Therefore, as Prototype Fund we were happy to fund a project which built a game on how to use Github while playing a game. The project is called **OhmyGit!** The game can be found and downloaded here: https://ohmygit.org/. This is the link to the demoweek and to a video which explains how the game works: https://archive.demoweek.prototypefund.de/runde8/projects/01-oh-my-git.html

There are many hosted services which make the use of Git and its features more intuitive. One of the most used ones is Github. However, since 2018 Github is part of Microsoft and hence not open source anymore. There are several other open source platforms that can be used instead of Github. Some examples are Gitlab, Gitolite, Gogs, Bitbucket and Codeberg. Moreover, as said before, Git can also be used alone.

The Prototype Fund also funded an open source alternative called **Gitea.** Here is a link to their website : https://gitea.io/en-us/.

Main source for the content: https://www.youtube.com/watch?v=DVRQoVRzMIY

□□ Linksammlung Einführung Git

- Git Tutorial (English - 43 minutes): https://www.youtube.com/watch?v=DVRQoVRzMIY
- Git Tutorial (German -13 minutes): https://www.youtube.com/watch?v=MgnRFZJ7M2s
- Git, Github and Github desktop (English - 23 minutes): https://www.youtube.com/watch?v=8Dd7KRpKeaE&t=950s
- Oh my Git! (German - 17 minutes): https://www.youtube.com/watch?v=fdPBfiWR0HY
- Oh my Git! (English - 3 hours): https://www.youtube.com/watch?v=dnAy6aIbIWo
- Open source alternatives to Github: https://opensource.com/article/20/11/open-source-alternatives-github

In fünf Schritten zur barrierefreien Software

Barrierefreiheit ist unverzichtbar - auch in der digitalen Welt. Sie ermöglicht es Menschen mit unterschiedlichen Behinderungen und temporären Einschränkungen, Technologien problemlos und in einer Vielzahl von verschiedenen Situationen einsetzen zu können. Open Source-Software spielt eine zentrale Rolle in unserer digitalen Gesellschaft, und als ihre Entwickler*innen haben wir die Verantwortung, diese für alle zugänglich zu gestalten. Ein Einstieg in die Thematik wird oftmals als kompliziert und aufwendig wahrgenommen. Dieser Artikel soll die Basics der digitalen Barrierefreiheit erklären und zeigen, dass schon einfache Anpassungen innerhalb von wenigen Stunden einen nennenswerten Fortschritt bedeuten können. Ein grobes Verständnis der Thematik ist für alle Entwickler*innen wichtig, um die Bedürfnisse von Menschen mit Behinderungen zu verstehen und zu berücksichtigen.Bei der digitalen Barrierefreiheit geht es hauptsächlich darum, wie Menschen mit Software interagieren. Dabei ist es egal, ob das über eine grafische Oberfläche, eine Website oder ein Command Line Interface (CLI) geschieht. Die Interaktion dort kann über verschiedene Eingabegeräte wie Tastatur, Maus, Touchscreen oder sogar per Sprache passieren. Die Ausgabe kann über einen Bildschirm, ein taktiles Braille-Display oder per Sprachausgabe mit einem Screenreader erfolgen. Eine Software muss so gestaltet sein, dass sie mit all diesen Eingabegeräten und Ausgabemedien funktioniert und gleichzeitig möglichst einfach zu verwenden ist. Dabei sind alle diese Geräte und Oberflächen nicht per se mehr oder weniger barrierefrei. Es kommt dabei immer und vor allem auf die Software und auch deren Nutzer*innen an.

Schritt 1: Verstehen, was Barrierefreiheit bedeutet

Barrierefreiheit in der Softwareentwicklung bedeutet, dass alle Menschen, unabhängig von ihren physischen, sensorischen oder kognitiven Fähigkeiten, Anwendungen problemlos nutzen können. Das ist eine sehr allgemeine Definition, die sich auf alle Arten von Software anwenden lässt. Verschiedene Menschen bevorzugen unterschiedliche Ein- und Ausgabegeräte oder sind wegen einer Behinderung sogar auf diese angewiesen. Dabei sind einige Methoden deutlich weniger flexibel als andere.

Ein Beispiel: Auf einem Bildschirm können viele Informationen gleichzeitig dargestellt und somit überflogen werden, während per Sprachausgabe immer nur ein Element nach dem anderen ausgegeben werden kann. Per Touchscreen können die meisten Menschen sehr schnell und präzise auf einen Button klicken, aber per Tastatur müssen sie erst den Fokus durch die gesamte

Oberfläche navigieren, bis sie ihn schließlich erreichen. Über ein CLI können erfahrene Nutzer*innen sehr schnell und effizient arbeiten, aber für weniger Versierte ist es oft sehr schwierig, sich die verfügbaren Kommandos und Abstraktionen zu merken - sie würden in nahezu allen Fällen eine GUI bevorzugen.

Übung: Prüfe, welche Interfaces für deine Software zur Verfügung stehen und wer genau dessen Zielgruppe ist.

Schritt 2: Verstehen, wie Hilfsmittel funktionieren

Die meisten Menschen bevorzugen einen weniger abstrakten Umgang mit Software: Angezeigt zu bekommen, welche Möglichkeiten sie haben und bei der Nutzung geführt und angeleitet zu werden. So müssen sie sich keine Shortcuts, Kommandos oder Abstraktionen merken, sondern können sich auf die eigentliche Aufgabe konzentrieren. Das ist auch - und gerade für - Menschen mit Behinderungen ein wichtiges Kriterium bei Software.

Viele von ihnen nutzen sehr spezifische Hilfsmittel mit ihrer eigenen, persönlichen Konfiguration. Eine Software darauf zu optimieren, ist sehr anspruchsvoll. Deshalb lohnt es sich, einen Blick darauf zu werfen, wie diese funktionieren. Nutzer*innen von Sprachausgabe navigieren beispielsweise mit der Tastatur oder verschiedenen Finger-Gesten zwischen den Elementen in einer Oberfläche, die sie dann jeweils vorgelesen bekommen. Dabei können sie ein User Interface anhand bestimmter Merkmale schneller erfassen und erkunden. Diese Merkmale könnten alles sein - Hauptsache ist, sie sind eindeutig und konsistent: Überschriften, Links, Buttons, Textfelder, Bilder, Tabellen, Formulare etc. Die Navigation per Überschriften erlaubt beispielsweise, sich schnell einen Überblick über den Inhalt eines langen Textes zu verschaffen, den andere Menschen einfach überfliegen könnten. Die Navigation per Buttons erlaubt es, schneller eine bestimmte Funktionalität der Software aufrufen zu können, die andere Menschen einfach nur mühelos auf dem Display antippen müssten. Gäbe es für eine Funktionalität eine Tastenkombination, wäre das für erfahrene Nutzer*innen eine weitere sehr schnelle Möglichkeit, diese aufzurufen. Diese könnte dann beispielsweise auch mit einem individuellen Makro ausgelöst werden, was den Einsatz der Software in vielen Bereichen deutlich flexibler machen könnte - auch für Menschen ohne Behinderung.

Nicht nur Menschen mit Sehbehinderung nutzen diese Art der Navigation sondern auch viele Menschen mit motorischen Behinderungen. Sie können beispielsweise nur sehr schwer oder gar nicht mit der Maus oder Touchscreens arbeiten und sind auf die Tastatur angewiesen. Aber auch diese könnte eine zu große Präzision erfordern, weshalb einige Menschen speziellere Eingabegeräte verwenden, die auf ihre eigene Behinderung angepasst wurden. Um nur mal ein paar der Möglichkeiten zu nennen: Eye Tracking, Head Tracking, Sprach-Erkennung, Joysticks, wenige große physische Tasten, die Controller von Spielkonsolen oder ein Saug-Blase-Schalter.

Viele haben gemeinsam, dass sie noch unflexibler sind als eine Tastatur und nur eine begrenzte Anzahl an verschiedenen Inputs erlauben. Ähnlich wie Nutzer*innen von Sprachausgabe-Software sind die Nutzer*innen dieser Hilfsmittel auf eine klare und eindeutige Struktur angewiesen, um sich in einer Software zurechtzufinden, da auch sie hauptsächlich linear und Element für Element einzeln navigieren müssen, um ein Interface zu bedienen. Aber: Wenn die Bedienung einer Software mit einer Tastatur möglich ist, dann stehen die Chancen gut, dass sie auch mit diesen Hilfsmitteln funktionieren kann. Diese Annahme ersetzt natürlich aber nicht die langfristige Notwendigkeit, die Software auch mit diesen Hilfsmitteln von echten Nutzer*innen testen zu lassen.

Es gibt aber auch noch andere assistive Technologien wie Screen Magnifier, die den Bildschirminhalt vergrößern, Kontrast-Anpassungen vornehmen oder Farben dynamisch umwandeln können. Diese sind für Menschen mit Sehbehinderung sehr hilfreich, aber auch für Menschen mit kognitiven Behinderungen, die sich beispielsweise besser auf eine Aufgabe konzentrieren können, wenn sie nur einen kleinen Ausschnitt des Bildschirms sehen. Auch hier ist es wichtig, dass die Software so gestaltet ist, dass sie mit diesen Technologien funktioniert.

Übung: Prüfe, ob die Bedienung deiner Software per Tastatur problemlos möglich ist und ob interaktive Elemente in einer sinnvollen Reihenfolge angeordnet sind. Falls nicht, kannst du hier erste Anpassungen vornehmen.

Schritt 3: Verstehen, was die Prinzipien der Barrierefreiheit sind

Auch die Prinzipien der Barrierefreiheit sind sehr allgemein gehalten und lassen sich auf alle Arten von Software anwenden. Sie sind sehr wichtig, um die Bedürfnisse von Menschen mit Behinderungen zu verstehen und zu berücksichtigen.

- Wahrnehmbarkeit: Alle Informationen und Bedienelemente müssen für alle Menschen wahrnehmbar sein. Das bedeutet, dass sie nicht nur sichtbar sondern auch hörbar oder fühlbar sein müssen. Dabei soll man auch sofort erkennen können, was die Funktion eines Elements ist und wie es verwendet werden kann. Zum Beispiel muss eine Checkbox auch aussehen wie eine Checkbox und nicht wie ein Button. Für einige Medien wie Bilder, Videos oder Audio-Dateien benötigen wir zudem eine Text-Alternative.
- Bedienbarkeit: Alle Bedienelemente sollen per Tastatur und assistiven Technologien erreichbar sein und intuitiv nach bekannten Regeln funktionieren. Zum Beispiel muss es möglich sein, in Drop-Down Menüs per Texteingabe oder Pfeiltasten zu navigieren. Wir müssen auch beachten, dass einige Menschen mehr Zeit benötigen und dass ihre Reaktionszeit unter Umständen deutlich länger ist.
- Verständlichkeit: Sowohl die Bedienelemente als auch die Informationen müssen verständlich sein. Das bedeutet, dass sie in einer klaren und einfachen Sprache formuliert sein müssen. Zum Beispiel könnte ein Button in einem Speichern-Dialog mit "Speichern" beschriftet sein statt mit "OK". Wenn Nutzer*innen Fehler machen, müssen sie auch

verstehen können, was sie falsch gemacht haben und wie sie eventuelle Probleme beheben können.
- Robustheit: Wir müssen sicherstellen, dass die Software für aktuelle und auch zukünftige Nutzer*innengruppen mit verschiedenen assistiven Technologien funktioniert. Das bedeutet, dass wir uns an die Standards halten sollten, die für die jeweilige Technologie gelten.

In den **Web Content Accessibility Guidelines** (WCAG) des W3C gibt es für alle diese Prinzipien sehr detaillierte Richtlinien, die auch konkrete Beispiele und Prüfkriterien enthalten. Diese sind sehr hilfreich, um die Bedürfnisse von Menschen mit Behinderungen besser zu verstehen und zu berücksichtigen. Sie lassen sich auch auf andere Arten von Software anwenden, die keine Web-Technologien einsetzen.

Die WCAG sind ein sehr guter Standard, der inzwischen in die Gesetzgebung in vielen Ländern eingeflossen ist. Wer sich längerfristig mit Barrierefreiheit auseinandersetzen möchte, sollte Zeit investieren, damit vertraut zu werden.

Aber nicht nur die WCAG ist ein wichtiger Standard, sondern auch die Dokumentationen und Standards der Betriebssysteme und GUI-Frameworks, die verwendet werden. Diese enthalten hilfreiche Informationen, wie Software barrierefrei gestaltet werden kann und welche technische Unterstützung notwendig ist, um gute Ergebnisse zu erzielen. Für Web-Anwendungen gibt es beispielsweise die WAI-ARIA Spezifikation, die eine Reihe von Rollen, Eigenschaften und Zuständen definiert, die für die Barrierefreiheit von Web-Anwendungen wichtig sind und assistive Technologien mit zusätzlichen Informationen über die Struktur einer Bedienoberfläche versorgt. Ein Tipp: Die meisten Vanilla-Komponenten von HTML5 oder beispielsweise SwiftUI sind bereits technisch barrierefrei und es empfiehlt sich daher immer, diese zu verwenden, statt eigene zu entwickeln.

Übung: Prüfe oberflächlich, ob deine Software den 4 Prinzipien der Barrierefreiheit entspricht. Falls nicht, kannst du mit den WCAG-Richtlinien genauer prüfen, welchen Verbesserungsbedarf es gibt und die entsprechenden Veränderungen vornehmen. Mit der Zeit wirst du ein Gefühl dafür entwickeln und die Prinzipien automatisch berücksichtigen, wenn du Software entwickelst. Der spätere Aufwand für Barrierefreiheit wird dadurch deutlich geringer.

Schritt 4: Implementierung und Test der Barrierefreiheit

Die Implementierung von Barrierefreiheit ist sehr vielfältig und hängt stark von der Art der Software ab. Es ist ratsam, sich mit der Dokumentation der verwendeten Technologien vertraut zu machen und die entsprechenden Richtlinien zur Barrierefreiheit zu befolgen. Dabei gilt es immer im Auge zu behalten, dass es nicht nur um die technische Umsetzung von Standards geht, sondern vor allem darum, wie echte Nutzer*innen die Software am Ende tatsächlich verwenden. Wenn

möglich sollte man Menschen mit Behinderung an der Entwicklung beteiligen - so fallen viele Barrieren schon früh auf und können direkt behoben werden, noch bevor die Software einem genauen Test unterzogen wird. Es bietet sich an, mit jedem Code Review kurz zu überprüfen, ob die Prinzipien der Barrierefreiheit eingehalten wurden und falls nicht, entweder eine Nachbesserung oder eine Erklärung einzufordern.

Der Test der Barrierefreiheit sollte immer von dazu speziell dazu ausgebildeten Menschen erfolgen, idealerweise auch wieder von solchen, die aufgrund einer Behinderung im täglichen Leben selbst auf assistive Technologien angewiesen sind. Dazu können beispielsweise auch die Prüfkriterien der WCAG verwendet werden, die sehr gut für die Prüfung von Software außerhalb des Webs geeignet sind. Ebenfalls empfehlenswert ist ein Selbsttest mit Screenreader-Software und Sprachausgabe. Für Windows gibt es den kostenlosen und Open Source-Screenreader NVDA, auf Linux gibt es Orca und auf macOS ist bereits VoiceOver vorinstalliert. Es bietet sich auch ein Test mit den etwas einfacheren, vorinstallierten Screenreadern auf den mobilen Plattformen iOS (VoiceOver) und Android (TalkBack) an. In beiden Fällen sollte man den sogenannten Bildschirmvorhang aktivieren oder den Bildschirm ganz ausschalten, um die Software nur per Sprachausgabe bedienen zu können. Ist die Software auch dann noch gut und einfach nutzbar, ist das ein sehr gutes Zeichen dafür, dass auch andere assistive Technologien ähnlich gut oder sogar noch besser funktionieren werden.

Die Bedienung von Screenreadern verlangt einiges an Übung und es ist am besten, wenn man dabei von einer Person angeleitet wird, die regelmäßig auf diese Software angewiesen ist. Das temporäre Verwenden von Screenreadern zeigt nicht, wie gut die Software für Menschen mit Sehbehinderung nutzbar ist, sondern nur, wie gut sie für Menschen ohne Sehbehinderung, aber dafür mit technischem Hintergrund nutzbar ist, die gerade einen Screenreader verwenden.

Zur weiteren Unterstützung bei der Integration von Barrierefreiheitsprüfungen in den Entwicklungsprozess gibt es auch einige Tools, die automatisierte Tests durchführen können. Diese sind aber ausschließlich Hilfsmittel und können die Prüfung durch echte Nutzer*innen nicht ersetzen. Ebenso könnten sie falsche Ergebnisse liefern, deren "Behebung" dann zu einer Verschlechterung der Barrierefreiheit führen würde. Empfehlenswert sind beispielsweise das WAVE Accessibility Tool von WebAIM oder die Accessibility Insights von Microsoft.

Übung: Mache dich mit Screenreader-Software vertraut und teste deine Software damit. Falls du keine Möglichkeit hast, mit einer Person zu üben, die selbst Screenreader-Software verwendet, kannst du auch die NVDA User Guides oder das Handbuch des von dir verwendeten Screenreaders durcharbeiten. Wenn dir fehlende Element-Beschriftungen oder fehlende Text-Alternativen auffallen, kannst du diese direkt beheben. Falls du Probleme mit der Bedienung der Software feststellst, kannst du diese ebenfalls ausbessern oder zumindest dokumentieren.

Schritt 5: Barrierefreiheit kommunizieren

Neben der technischen Umsetzung der Barrierefreiheit ist es auch wichtig, diese zu kommunizieren. Das kann beispielsweise über eine Dokumentation, eine Website oder eine Release-Note geschehen. Es ist wichtig, dass die Nutzer*innen wissen, dass die Software barrierefrei ist, wo und ob es noch Probleme gibt, und wie sie diese melden können. Barrierefreiheit sollte dabei stets einen prominenten Platz einnehmen und nicht nur als Randnotiz erwähnt werden. Release Notes wie "Verbesserte Barrierefreiheit" sind nicht hilfreich, da sie nicht erklären, was genau verbessert wurde und wie sich das auf die Nutzer*innen auswirkt. Oftmals sind Menschen mit Behinderung von neuer Software eher abgeschreckt, da sie befürchten, dass diese nicht barrierefrei sein könnte. Es ist daher wichtig, dass sie wissen, dass die Software barrierefrei ist und dass sie sich darauf verlassen können, dass sie auch in Zukunft barrierefrei bleiben wird.

In der Kommunikation um dieses Thema gilt es außerdem, immer absolut ehrlich zu sein und typische Marketing-Relativierungen zu vermeiden. Behinderte Nutzer*innen sind sehr gut darin, diese zu durchschauen und werden sich dadurch eher abgeschreckt fühlen. Es ist hier besser, Probleme offen und detailliert zu kommunizieren, da dies selbst eine Verringerung der Barrieren darstellen kann. So sind die Nutzenden dazu in der Lage, selbst zu entscheiden, ob sie die Software aktuell verwenden können oder nicht.

Auch in der Kommunikation müssen die Prinzipien der Barrierefreiheit beachtet werden. Das bedeutet konkret: Informationen klar und einfach formuliert sein. Ebenso müssen sie leicht auffindbar sein und dürfen nicht versteckt werden. Sie müssen selbst barrierefrei sein, damit sie auch von Behinderten gelesen werden können und sie müssen sich immer auch auf den aktuellen Stand der Software beziehen.

Sind diese Voraussetzungen erfüllt, werden auch Personen, die assistive Technologien einsetzen, früher oder später auf die Software aufmerksam und werden sie ausprobieren. Das ist eine sehr gute Gelegenheit, um qualitatives Feedback zu erhalten und die Software weiter zu verbessern - für alle Nutzer*innen.

Übung: Erstelle eine "Erklärung zur Barrierefreiheit" für dein Projekt und erkläre darin, wie du die Barrierefreiheit sicherstellst, welche Barrieren es derzeit gibt und bis wann diese behoben werden sollen.

Fazit

Digitale Barrierefreiheit ist nicht nur ein Konzept oder eine Anforderung - sie ist ein Wegbereiter für Gleichberechtigung und gesellschaftliche Teilhabe. Durch die Schaffung barrierefreier Software

ermöglichen wir Menschen mit Behinderungen, ein großes Maß an Selbstbestimmung zu erlangen und aktiv an unserer Gesellschaft teilzunehmen. Sie können so kommunizieren, Informationen suchen, Dienstleistungen nutzen, sich weiterbilden und ihrer Arbeit nachgehen - all das mit der gleichen Leichtigkeit und Bequemlichkeit wie alle anderen auch.

Es stimmt: Die Umsetzung von Barrierefreiheit in der Softwareentwicklung kann eine große, zusätzliche Herausforderung darstellen. Es gibt nicht *die eine Checkliste*, sondern es erfordert ein Verständnis für die Bedürfnisse verschiedener Benutzer*innengruppen, ein Umdenken in Design- und Entwicklungsprozessen und den Einsatz zusätzlicher Tools und Tests. Aber die zusätzliche Arbeit, die in die Schaffung barrierefreier Software investiert wird, zahlt sich auf vielfältige Weisen aus.

Erstens verbessert es die Erfahrung für alle, nicht nur für Menschen mit Behinderungen. Features, die zur Barrierefreiheit beitragen, wie zum Beispiel ein guter Kontrast, klare Navigation und einfache Sprache, sind eine Hilfe für alle.

Zweitens erweitert es den potenziellen Nutzer*innen-Kreis ihrer Software erheblich. Es gibt Milliarden von Menschen mit unterschiedlichen Arten von Behinderungen auf der Welt, und barrierefreie Software ist ein entscheidender Faktor, um diese Menschen zu erreichen und zu bedienen.

Drittens fördert es Innovation und Qualität. Wenn man sich darauf konzentriert, Software für alle zugänglich zu machen, wird man oft kreativere Lösungen finden und ein höheres Qualitätsniveau erreichen. Gerade Open Source-Software scheitert aktuell häufig daran, für die meisten Menschen außerhalb der Tech-Community attraktiv zu sein. Barrierefreiheit und User Experience können hier einen großen Unterschied machen.

Es gibt einige Schulungs-Angebote, unter anderem von Menschen mit Behinderung, die sich auf die digitale Barrierefreiheit spezialisiert haben. Diese sind sehr empfehlenswert, um weitere Verbesserungen der Barrierefreiheit zu erreichen und die eigene Software besser zu verstehen.

Also, liebe Entwickler*innen, lasst uns den Weg der digitalen Barrierefreiheit beschreiten. Es mag ein bisschen mehr Arbeit sein, aber es ist eine Arbeit, die sich lohnt. Es ist eine Arbeit, die den Unterschied macht - denn sie macht unsere Software, unsere Gemeinschaften und unsere Gesellschaft besser und inklusiver. Zusammen können wir eine digital zugängliche Welt schaffen, in der alle Menschen gleichermaßen teilhaben können. Und das ist ein Ziel, das jede Anstrengung wert ist.

Eine Übersicht der genannten Tool

- Die Prinzipien der Barrierefreiheit
  - Wahrnehmbarkeit

    - Bedienbarkeit
    - Verständlichkeit
    - Robustheit
- Web Content Accessibility Guidelines (WCAG) des W3C
- Für Web-Anwendungen die WAI-ARIA Spezifikation
- Für automatisierte Tests: das WAVE Accessibility Tool von WebAIM
- Für automatisierte Tests:Accessibility Insights von Microsoft.
- Screenreader Guidelines: NVDA User Guides

Zusatz: Eine schnelle Barrierefreiheitsüberprüfung für Websites bietet auch das Tool https://www.experte.de/barrierefreiheit.

Zur Autorin

Casey Kreer ist Beraterin für digitale Barrierefreiheit und Software-Entwicklerin. Sie ist seit ihrer Geburt sehbehindert und nutzt assistive Technologien, um mit dem Computer zu arbeiten. Sie arbeitet als Freiberuflerin für verschiedene Kund*innen und Projekte, die gute und inklusive Medien-Angebote schaffen möchten. Dafür vermittelt sie in Workshops und Schulungen das notwendige Wissen oder schreibt Gutachten über Barrieren in bestehender Software. (conesible.de)

□□ How to work with a mentor – Interview with Common Syllabi

Pierre Depaz and Tobias Schmidt developed Common Syllabi in the 11th round of the Prototype Fund. With Pierre being relatively new to the open-source universe he decided it'd be useful to work with a mentor for the project: Tobias, who has participated in a lot of open-source projects for quite some time. In this interview Pierre talks about the benefits of this model, what the mentorship looked like and gives some advice to people who are also looking for support.

While developing Common Syllabi you worked together with a mentor – why did you decide to do that?

When I started to think about the idea for Common Syllabi, I immediately thought about sustainability for the project: just because you build it, it does not mean that people will use it, and this gets less and less likely throughout the project lifespan. Common Syllabi is a platform to archive and share learning materials across universities. Because of the archival part, there needed to be particular attention paid to the sustainability of the project, both from a technical and organizational perspective.

I decided to work with Tobias as a mentor, because he already had experience in these things, having participated in large and successful open-source projects from the beginning. He could therefore bring some past experience and technical expertise regarding the reliability of the project and its accessibility. In the end, this was a great decision, and helped a lot to kick start the project on solid foundations.

Did you know each other before or did you start looking for a mentor once you knew you wanted to do the project?

Tobias and I knew each other from before. I met him as he was working on an open-source project, the Prometheus monitoring system, at his former company and was very interested in the organizational choices that can make such an open-source project sustainable, as well as in the whole technical process: how do you decide what to work on, how do you design the project from the start, who contributes and how, etc.

Coming from a political science background myself, and having just switched to game development, I knew a little bit about programming, and a little bit about open source, but only as an amateur/from an outsider perspective. That being said, we had already hacked on a few side

projects together, so we knew that we also got along when in a work situation.

To be honest, I would not know where to start if I had to find a mentor once the project had started, and the fact that I knew Tobi already gave me the confidence, that I could deliver what I would set out to during the funding phase.

What were the benefits from this mentorship for you? What would have been different without the mentor?

For me, the benefits were huge. Besides giving me confidence that I was not in this alone, I also learned a lot from a technical perspective: working with an experienced professional is invaluable. This involved coding practices, but also documentation, repository organization, devops, etc.

Tobias: for me, I just really appreciated being able to think about a project at the high level without having to spend time implementing technical details or doing menial work. This mentorship helped me develop more manager skills, as I help junior developers accomplish tasks at the company I am currently at. I also liked the opportunity to write some code in Go!

What did the mentorship look like?

Tobias was on the official application, but we made sure to discuss all of the practical details before we officially applied. This meant a couple of things: as little administration as possible for Tobias (since he already has a full time job, he could only commit a few hours a week), and bi-weekly meetings for checking in on progress and setting out subsequent goals, a little bit in the agile style of working.

The first meetings were focused on the architecture and overall design: I did a lot of the user research, and based on what I found of how people are using current tools, or how they would like to use future tools, we decided on a design that would be lightweight and reliable. The second part of the funding phase was on the same schedule, but rather about code and technical details of implementation. I would work on the design and write down some questions and issues that I encountered, and when we met we would do code reviews to see what worked, or what did not, and how to move forward.

In terms of advice, I would say that there are a few things that were important. First, ideally find someone that you also appreciate at the human level, which makes it a lot easier and fun to work with. Even if you meet someone specifically for this, it would be important to meet at least once before any official planning to see how the mutual feeling is. Second, in-person meetings are a lot faster and more productive than online, when it's about designing or debugging. We only did in-person meetings, which helped to cover a lot more topics and deal with digressions in a more pleasant manner. Third, I would say it's important that there has to be something that the mentor gets out of it as well, whether it's honing some skills, trying a new technology, or even just believing in the project; this helped to keep motivation and focus. Finally, an important part was, that we were both flexible to adapt to each other's schedules. All of this made sure this was remaining a fun and engaging project for both.

Do you have any other tips or ideas for people who want to do a project but are missing a certain skill in order to realize it?

Tips for applicants would be to (1) believe in the idea and (2) scope it down! Even if you don't have all the team members or skills to develop an idea, it's easier to get people on board once you have a clear idea (which the application process helps to formalize), and it's always possible to find some people for specific tasks along the funding phase.

That being said, it's also important to know what are the people/skills that you are missing: it's easy to deal with known unknowns, but more complicated with unknown unknowns (you don't know what you don't know, so it's hard to ask for help about it!). Doing a bit of research on OSS organization (such as browsing the OSAOS Handbook, or OpenCollective) helps a lot in figuring out the main roles in an OSS organization and in identifying where you need help.