Lost in Translation: Complexities and Good Practices of Translating Coding-Based Teaching Materials Across Multiple Human Languages

Olexandr Konovalov

School of Computer Science, University of St Andrews
obk1@st-andrews.ac.uk

Lucia Michielin

Edinburgh Futures Institute, University of Edinburgh
lucia.michielin@ed.ac.uk

Yanina Bellini Saibene

rOpenSc; RLadies+; Universidad Austral
yabellini@gmail.com

Introduction

Translating coding-based teaching material may look like a straightforward practice, especially if you are translating from English to your native language. However, this process is often undertaken in contexts where learners are already navigating coding concepts through a second language, which adds an additional layer of difficulty. If you are directly involved in teaching such students, or participate in international educational projects like The Carpentries[184] or Programming Historian[185], you may have considered translating teaching materials to make them more accessible.

Based on our experience in translations and localisation projects, we decided to write this short contribution to help others navigate similar situations and avoid common pitfalls.

This contribution is a collection of personal reflections and good enough practices we learned through the years, with the caveat that any translating project will have its own peculiarities and needs. While many of our reflections stem from medium to larger, collaborative efforts, some of these practices may also be adapted to smaller-scale projects, although this is not the primary focus of this chapter.

General Considerations

Before moving into the practical side of this contribution (organised around the different project stages: designing, deploying, and maintaining), we wanted to outline a few general considerations.

As mentioned earlier, it is often the case that translation occurs from English into another language that is the translator's native language (at least this was our shared experience), reflecting the dominant role of English in coding and academia. While this chapter focuses primarily on that direction, other forms of translation may be required in different contexts.

Translating from another language into English follows essentially the same processes and can rely on the same tooling as translation from English into other languages. It also requires establishing shared agreements within the community, just as in any other translation effort. For example, the rOpenSci translation guidelines[186] include language-specific chapters that document community decisions on how to approach translation into each language, highlighting that these agreements are integral to the workflow regardless of the direction of translation.

Moving from a language into one's native language may create a misleading sense of reassurance: if one was able to design and teach material in a language that is not their primary one, it might seem reasonable to assume that doing so in their mother tongue would be considerably easier. This is not always the case.

First of all, if you have studied a subject in a certain language (e.g. English), it is harder to then convey the core topic in a different one. Few situations are more frustrating than knowing how to refer to a concept in a second language but being unable to recall the equivalent term in one's native language, and this experience is likely to recur frequently throughout the process.

Because English has been the vernacular language for programming languages, their official documentation and most of the coding-based teaching material available on the internet, the word you are searching for may not actually have a clear translation into the other language. This brings us to another fundamental issue: how much should you actually translate? Where do you stop? The aim is to create material that is understandable for your audience, while also providing them

[184] https://carpentries.org/
[185] https://programminghistorian.org/
[186] https://translationguide.ropensci.org/

with enough background to refine and progress their knowledge beyond your course.

If you fully translate everything and introduce many neologisms, you risk preventing learners from efficiently searching for further information online. On the other hand, if you do not translate and explain key concepts and jargon sufficiently, students may struggle to fully comprehend the content. It is a fine line to navigate, made even more complex when dealing with non-Latin alphabet languages, where transliteration also becomes a concern.

No single solution fits all, and much will depend on the language in question. In collaborative projects, community discussion is often central to these decisions, making the process more collective and sustainable over time. As a rule of thumb, it is helpful to translate terms that can become established jargon in the target language, while leaving others in English. This can be supported through "double-barrelling" keywords, where both the original English term and a more accessible translation are retained.

More generally, any translation project benefits from two key approaches: a collaborative approach, involving people with diverse linguistic and subject expertise, and an iterative approach that allows materials to improve over time. The more the material is used, the more it can be refined. The wider uptake of the material is also a good indicator of a successful translation project, because the main aim of any translation programme is to reach a wider audience and increase overall literacy.

Having discussed these general approaches and issues, we now move into more targeted advice based on the different stages of the project. Although it may seem more applicable to a large and medium-sized translation workflows, you may find it useful to consider even when starting a small personal project, since it may be helpful to set up things in a scalable and reproducible manner from the start (in the same way like setting up a Git repository for a personal task).

Before Starting: The Design Stage

Approaching a translation project, it's important to have a clear idea about its organisation. You are going to face a substantial task. Choosing the wrong tools and process may cause substantial delays later or impose very restrictive limitations.

Internationalisation vs. Localisation

A common view is to separate internationalisation (commonly denoted as "i18n") and localisation ("l10n") steps[187]. Internationalisation is focussed on creating multi-language versions of the product (e.g. book, website, software, etc.), while localisation is usually a separate step which happens later.

In the context of training materials, for example, translation would use the same examples and datasets, as in the original version; an exception to this general rule is the usage of specific units of measure, currency and date format that may not make sense in a translated version. A localised version of the training materials would use variables names, examples and datasets that are more relevant to the language target audience. Separating these two stages makes the first one more achievable.

You also need to decide whether there is a real need in the localisation of examples balancing on student understanding and engagement and maintenance of the translation — if the English source is frequently updated, it will make it harder to keep the languages synchronised. Focusing on internationalisation (meaning translating the main text only) is often a more realistic goal than localisation (which includes translating and adapting more than the main text). Working through this process can help clarify your next steps.

Technical Tools

Next, you need to consider the technical infrastructure for your translation project. There are some aspects to keep in mind for ad hoc approaches without using any of the specialised platforms.

Several translation projects have used tools and roles that are already well known in the world of software development: the project takes place on GitHub or a similar version control platform, organized by GitHub projects where you can track the status of each task and who is responsible for it, where an initial machine translation is performed and presented as a Pull Request, which is then reviewed by usually 2 different people to correct common errors made by machine translators and ensure consistency in the translation in accordance with the agreements of the team. This flow replicates other processes with familiar roles (editors, reviewers) and tools (Markdown, version control), enabling more people to use it more quickly. In addition, those who are unfamiliar with the tool can be trained, which has the added benefit of teaching them a tool that will be very useful beyond the translation project.

A straightforward idea might be to use, for example, a web-based editor such as Google Docs, or, if the source is in Markdown format, a collaborative editor like HackMD, or, if you want to translate a website, just edit raw HTML files. Indeed, this seems to be a viable approach — the first two allow real-time collaboration and leaving comments; the latter two can be used with version control. Not that it is impossible — but it needs to be well thought through.

A potential problem is that it does not scale well, especially if the source is not static, and it is necessary to periodically update the translation to keep it in sync with the original. Another challenge is to maintain consistency of translation —

[187] See Wikipedia - Internationalization and localization

when multiple translators are working on different parts of the document, they may translate the same terminology in different ways. Finally, you want to track the progress and easily see which parts of the translated document are already finalised and which need discussion. All these tasks are typical for collaborative translation projects, and are helped by specialised purpose-built systems. Thus, from such a system you can expect a customisable workflow in which each fragment of text can be independently translated, proofread,

updated and approved, possibly by different persons involved at each step. In addition, there should be some semi-automated procedures (so you may have additional quality control steps) to export translations from the platform (i.e. to make them live on a website) and to pull new versions from the original source, indicating which portions of the text have been updated and need to be translated again. The latter operation may be assisted by suggested translations, based on the previous versions — this is what is called “Translation memory”, often abbreviated as TM.

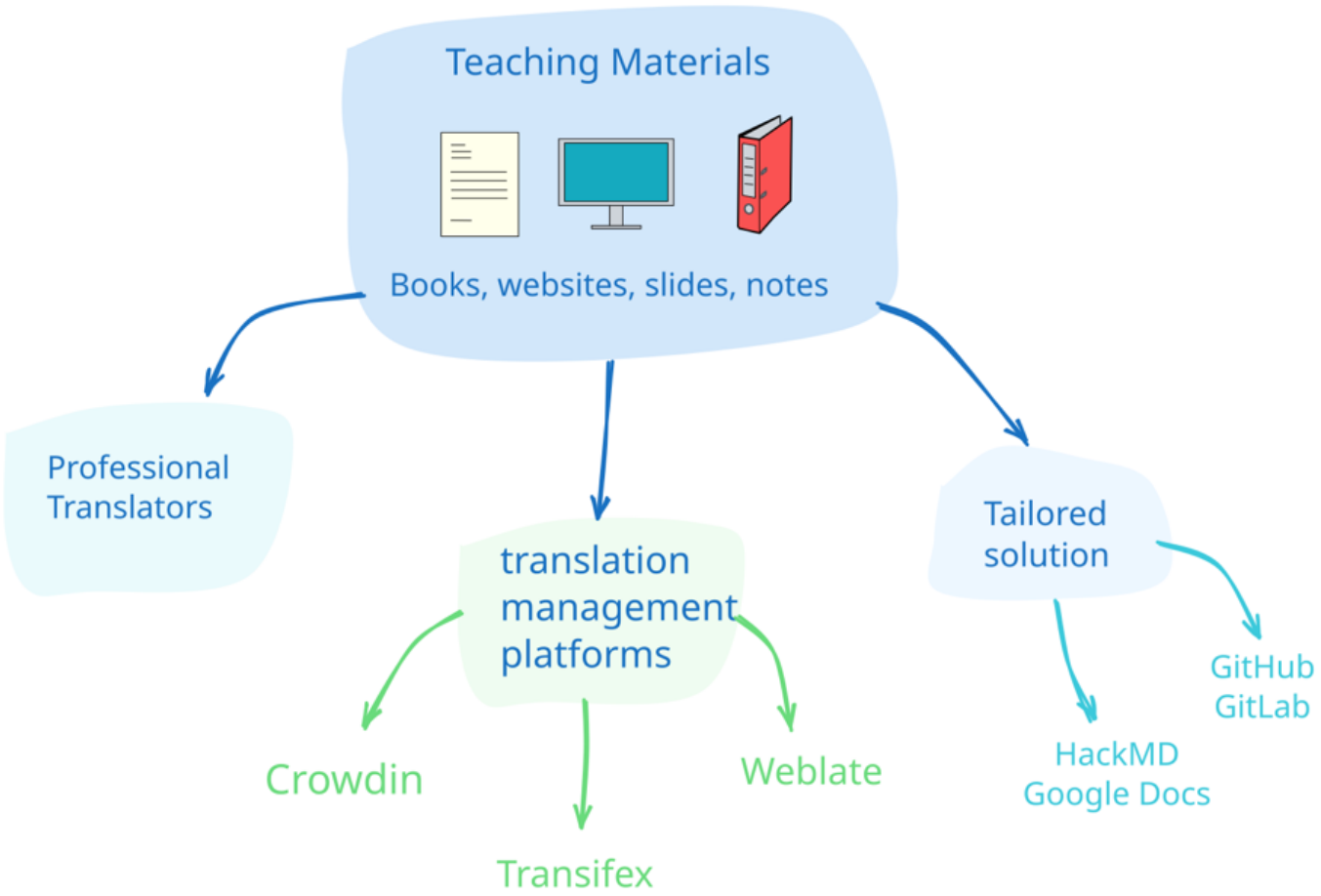


Figure 40.1: Different approaches to translating teaching materials: from professional services to platforms and customized workflows.

In the following table, we describe several existing platforms for collaborative distributed translation: Crowdin[188], Transifex[189], and Weblate[190]. It is not feasible to make a single recommendation because so much depends on the specificity of your project, but we tried to collect as much information as possible in order to make your choice easier.

Table 1: Comparing Platforms for Collaborative Translation

Feature:	**Platform:**		
	Crowdin	**Transifex**	**Weblate**
Type	Cloud, SaaS	Cloud, SaaS	Self-hosted, SaaS
Open source	No	No	Yes
Key integrations	GitHub, GitLab, Bitbucket, Figma, Slack, Jira, API	GitHub, GitLab, Bitbucket, API, Slack, Jira	GitHub, GitLab, Bitbucket, API, REST hooks, custom scripts
Supported content formats	60+ (PO, XLIFF, JSON, YAML, etc.)	50+ (PO, XLIFF, JSON, YAML, etc.)	60+ (PO, XLIFF, JSON, YAML, etc.)
Collaboration	Web UI, comments, tasks, roles	Web UI, comments, tasks, roles	Web UI, comments, review queue
Glossary & Translation Memory	Yes (Advanced)	Yes (Advanced)	Yes (Basic, extensible)
Machine translation	Google, DeepL, Microsoft, custom	Google, DeepL, Microsoft	Google, DeepL, Microsoft, others
Pricing	Subscription, custom plans. Limited free tier (open source projects: free)	Subscription, custom plans. Limited free tier (open source projects: free)	Free (self-hosted, free), SaaS plans (free tier for open source projects)
API/CLI	REST API, CLI	REST API, CLI	REST API, CLI
Mobile SDK	Yes (iOS/Android)	Yes (iOS/Android)	No
Quality checks	Yes (customisable)	Yes (customisable)	Yes (customisable, scriptable)
Workflow Automation	Advanced	Advanced	Highly customisable
Community Projects	Supported (OSS discounts)	Supported (OSS discounts)	Native support
Used by	The Carpentries, The Turing Way	Localization Lab, Open edX	The R-Dev Group

[188] https://crowdin.com/ | [189] https://www.transifex.com/ | [190] https://weblate.org

The Implementation Stage: The Translation Workflow

When it comes to the deployment stage, much will depend on the specific tools and setup you choose to use. However, we have gathered a series of good practices and common pitfalls that should help streamline the deployment phase, whatever your workflow. The following tips are designed to support a smoother, more efficient translation workflow as you bring your teaching materials into multiple languages.

Peer Review

Whenever the translations are made in a specialised platform like one from the table above, submitted via GitHub or GitLab pull requests, or contributed in some other way, it is crucial to have them reviewed. This helps to ensure consistency of translation across the project and to propagate the knowledge within the translation team. Make sure that you provide feedback to the translators, instead of quietly correcting their errors. Timely feedback is especially important when new translators join the team: delayed feedback may result in them repeatedly making the same errors, causing more work to fix them in the future.

Reviewing collaboratively-made translations imposes the need for an approval stage (for example, a button in the web interface). With some reasonable exceptions (e.g. a code example that needs no translation), we recommend following the rule that one should never approve one's own translations or corrections. Instead, consider a ping-pong-like exchange between the translators. When one of them makes a pass over a section of a document, the other approves it or makes corrections and returns it for approval or revision by the original translator. The tools should help you organise such a workflow in a distributed setting.

For languages spoken in multiple countries, revisions should ideally involve reviewers from different regions. For example, in Spanish, it is beneficial to include reviewers from several countries to ensure that the chosen words and expressions are widely understood. Don't underestimate that asynchronous communication will take more time. Sometimes getting together in an online meeting helps to traverse and finalise the remaining fragments of the text much faster and get the next portion of the document ready for publication.

Onboarding New Translators

Onboarding new translators is a regular task in community-driven projects. It is important to prepare guidance and provide training for new contributors. Here, there are a few things that need to be explained:

- **Is it important for your project to provide the exact word-by-word translation, or can one rephrase the text freely to convey the meaning?** When translating a coding-based teaching material, it might be sensible to use a completely different phrase or definition that will be more understandable to the target audience. Doing this, one should find the right balance — diverting too much from the original may lead to inconsistencies with other parts of the material, and may be harder to maintain in the future if the original text changes.
- **Are there any parts of the text that should not be translated?** The source may have a specific format and may include parts used to build the resulting document or website, and these should remain intact. New translators might be unfamiliar with this format and accidentally translate too much. Prevent unnecessary effort by explaining things like specific markup constructions that should not be translated, hyperlinks, cross-references, etc. Explain also whether it is important to preserve formatting and try to approximate line breaks.
- **How to communicate efficiently?** For example, how to reach a user within the collaboration platform (e.g. referring to their username), and how to set up your notifications to avoid missing them. Setting up communication channels efficiently would allow to keep the discussions closed to the translations (e.g. as comments on the pull request or in the translation management system). In addition, consider having a live channel for interactive discussions and technical support (e.g. Zulip or Slack).
- **Which parts of the document are prioritised for translation?** If someone joins in the middle of the project, are there any parts already translated which are recommended to read first? Also, consider a workflow which allows publishing partial translation, for example, the first few sections of the teaching materials. This may allow you to run pilot trainings, and may attract more translators to your project.
- **How to submit improvements to already-made, and possibly even approved translations?**

Glossary

Finally, if the glossary is not a part of the translated document, we recommend establishing it. It could at least be used by the translators; even better, you might consider adding it as an appendix to the translated version. Another useful tip is to add English terms in parentheses when a notion is introduced in the text for the first time.

The Long-Term Maintenance: General Good Practices

While the creation and deployment of multilingual teaching materials are critical steps, it is equally important to recognise that the process does not end there. Digital and web-based resources are inherently dynamic — they need periodic updates as the software changes, educational needs evolve, or the content becomes outdated. Over time, links may break, interfaces may change, and new technologies might require updates to ensure continued accessibility and functionality.

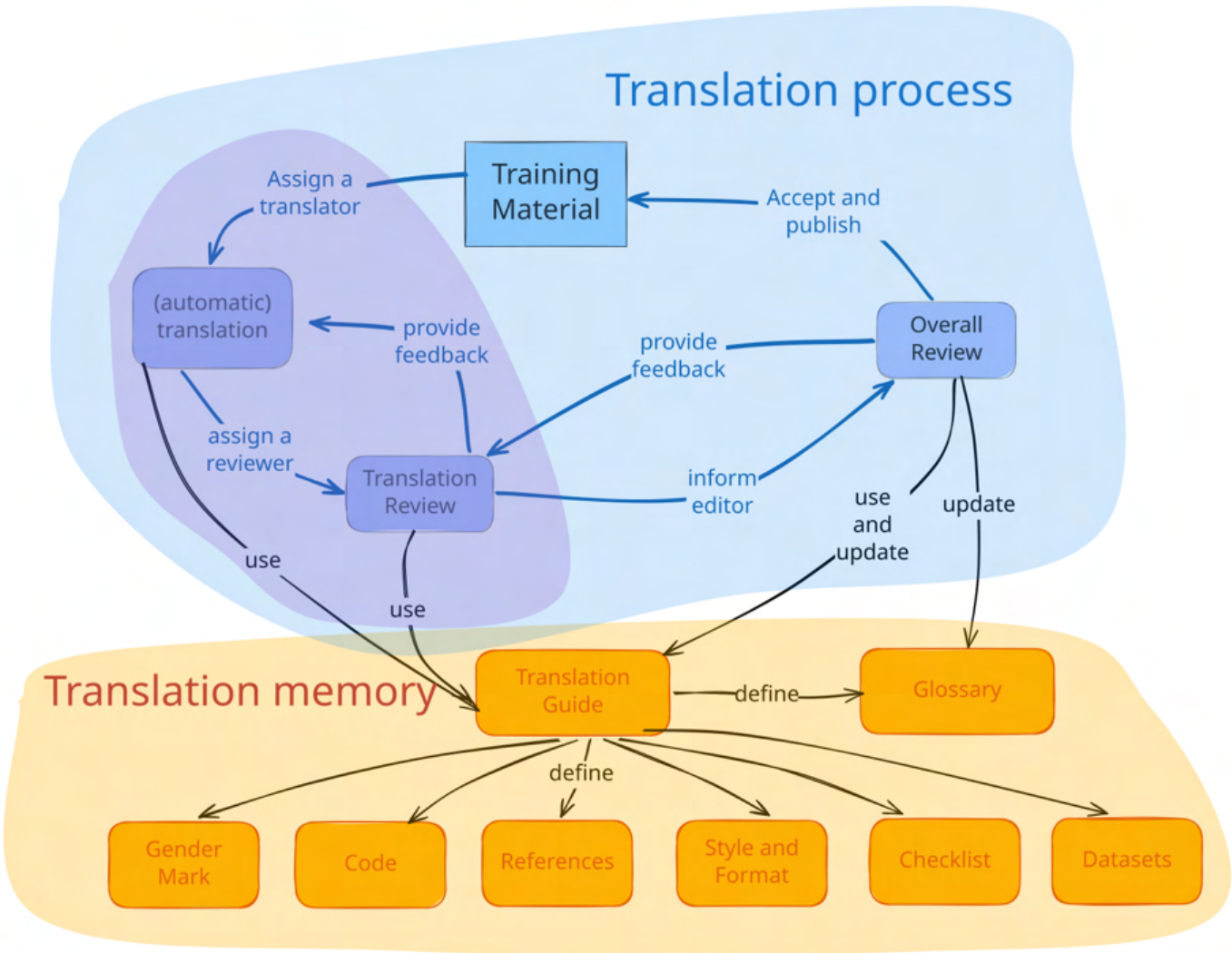
Long-term maintenance, therefore, must be part of your project planning from the very beginning. This means allocating both financial resources and sufficient staff time for ongoing review, updating, and technical support. Consider who will be responsible for regularly checking the content, responding to user feedback, and implementing updates across all language versions. Budget not only for immediate project costs, but for the recurrent, perhaps less visible, work required to keep your materials accurate and effective.

A proactive approach to maintenance prevents issues from accumulating unnoticed, minimises disruption to users, and helps ensure that your investment in high-quality, inclusive teaching resources continues to deliver value over time. With this in mind, we have collected some general good practice advice to help establish sustainable maintenance practices for your multilingual materials.

Use version control! If your workflow is based on contributing translations via GitHub or GitLab pull requests, you are already there. This gives you reviewer functionality and access to other tools from their ecosystem, such as publishing static websites or uploading releases on Zenodo. If you use a specialised translation management platform, the best approach is to use version control anyway. The input and output text may be stored in the

outside the platform, and the platform's integration with the repository hosting service would allow a semi-automated import of the original version and exporting the translation. Using the external repository would reduce the dependency on any particular platform. Remember to keep a record of who worked on the project as translator and reviewer to give them credit — this information will be automatically recorded if you are using GitHub or GitLab, but otherwise you may need to take note of it manually.

Figure 40.2: Translation workflow for training materials, showing how translation, review, and feedback interact with supporting resources like translation guides and glossaries, and style conventions.



It may also happen that while you are working on the translation, a new version of the original text is released. Importing updates into your project may set you back. It might be useful to think of versions of the documents, first publish a full translation of some version, and then import and work on the new one, instead of trying to sync two moving targets, and at no single point having a full translation. This, however, very much depends on the particular project. If it is more important that the document is up-to-date than fully translated, using the specialised platform, you may set up a workflow that renders a document with translated and approved fragments in the target language, and the rest in the original.

Finally, translating code-based teaching materials is a challenge: one should be fluent in the terminology and subtleties in both languages and understand the technical content. Welcoming contributors who do not necessarily possess all three skills, but just one, and are willing to learn through collaboration and feedback, would make your community more inclusive and diverse.

Final Remarks

Language shapes who can access knowledge, participate, and contribute in communities. When most resources are available only in English, a large portion of the global community faces structural barriers that cannot be overcome by individual effort alone. Addressing these barriers is essential not only for being fair but also for the sustainability and global relevance of scientific and open-source initiatives and projects.

We wrote this text based on the experience and/or recommendations from The Carpentries[191] and the Ukrainian Carpentries Subcommunity[192], The Turing Way[193], "Teaching Tech Together" (Enseñar tecnología en comunidad)[194], rOpenSci Multilingual Publishing Project[195], and "R for Data Science" (R Para Ciencia de Datos)[196].

If you wish to explore more hands-on examples of translation projects, you can have a look at the following:

- Multilingual Data Science[197] is a blog by Yanina Bellini Saibene, where she explores her own experience in creating multilingual data science teaching material.
- The documentation for The Turing Way translation workflow[198].
- The documentation for the internationalisation of The Carpentries lessons[199].
- An example of guidance for translators[200] from the Ukrainian Carpentries subcommunity.
- A multi-lingual glossary[201] for computing and data science terms with a GitHub-based contributing workflow.
- The rOpenSci Localisation and Translation Guidelines[202] explain why and how they localize and translate their resources.
- The guidelines for translations of Programming Historian material[203].

Through translation initiatives, we can help lower these barriers by expanding access to knowledge and participation. By creating technical and social infrastructure for multilingual resources and recognising translation as a meaningful form of contribution, we can foster more inclusive and diverse communities and enable more people to engage, contribute, and shape open source projects.

[191] https://carpentries.org/
[192] https://ukrainian-carpentries.github.io/
[193] https://book.the-turing-way.org/
[194] https://teachtogether.tech/es/
[195] https://ropensci.org/multilingual-publishing/
[196] https://es.r4ds.hadley.nz/
[197] https://datasciencebydesign.org/blog/multilingual-data-science
[198] https://github.com/TWTranslation/translation_workflow
[199] https://github.com/joelnitta/carp-i18n-guide
[200] https://github.com/ukrainian-carpentries/translation-guide
[201] https://github.com/carpentries/glosario
[202] https://translationguide.ropensci.org/
[203] https://programminghistorian.org/en/translator-guidelines