Preparing an R Package for Open-Source Contributions: An Experience Report on the World Wildlife Fund's Forest Foresight

Amin Bakhshi[a], Hasrul Maruf[a], Maas van Apeldoorn[a], Zillah Calle[b], Jonas van Duijvenbode[b], Ismay Wolff[a], Yanja Dajsuren[a] and Jacob Krüger[a]

[a]*Eindhoven University of Technology, The Netherlands*
[b]*World Wildlife Fund Netherlands, The Netherlands*

ARTICLE INFO

Keywords:
Software Evolution
Community-Driven Contributions
Repository Management
Open-Source Development

ABSTRACT

Deforestation (i.e., the removal or destruction of forests by humans), particularly illegal, is a major cause of ecological and environmental problems. To combat illegal deforestation, the World Wildlife Fund (WWF) has developed an open-source R package to predict deforestation around the world using machine learning. The package has been used by and customized to various countries, providing immense value for environmental protection. However, the package was implemented by domain experts without software engineering background, resulting in an unstructured development process, a monolithic codebase, and a lack of documentation on processes and code. Aiming to build an open-source community to improve and maintain the package, the WWF team decided to focus on enhancing the accessibility and attractiveness of the codebase for newcomers. Supporting this goal, we conducted an action-research-like project using Scrum to improve the code quality, tooling, testing, processes, and documentation while also establishing practices to sustain and build upon these improvements. In this article, we describe this project and share our insights into opening an R package to make it more accessible for external open-source contributors. Our insights include guidance on communicating design decisions to domain experts without a software engineering background and on how to train them in software engineering practices. Further insights highlight the specific challenges of working with R packages. Lastly, our work showcases the contributions that software engineering can make to support environmental protection and can guide future projects in this direction.

1. Introduction

Deforestation is a serious threat to the environment, the climate, and humans. Between 1990 and 2020, 420 million hectares of forest have been lost globally (FAO, 2020). Although the rate of deforestation is on a downward trend, between 2015 and 2020, approximately 10 million hectares of forests have still disappeared annually. Unfortunately, even if the decline in forest loss continues, it is unlikely that the goal of the 2021 United Nations Climate Change Conference (COP26) of ending deforestation by 2030 will be met (Einhorn and Buckley, 2021; FAO, 2020). The consequences of current deforestation threaten more than 100,000 species, release 6.7 billion tons of CO^2 annually (15 to 20 % of global emissions), alter rainfall, and increase soil erosion; leading to natural disasters (e.g., floods, droughts) and millions of people being directly or indirectly impacted globally (WWF, 2022; FAO, 2020; Ripple et al., 2024; Wolff et al., 2021; Bologna and Aquino, 2020).[1]

Interpol (2021) estimated that around 15 to 30 % of global timber production comes from illegal logging. Similarly, the *International Consortium of Investigative Journalists* repeatedly reports on severe cases of illegal deforestation around the world under the "Deforestation Inc." initiative.[2] To combat such illegal deforestation, the World Wildlife Fund (WWF) is developing an open-source R package called *Forest Foresight*.[3] Forest Foresight is designed to enable users (e.g., governments, businesses, local communities, police) to predict illegal deforestation (WWF, 2022). To obtain reliable predictions, Forest Foresight uses machine learning and near-real-time monitoring.

Forest Foresight was initially developed by two WWF domain experts (fourth and fifth author) and external consultants without formal software engineering training. At times, this has resulted in the development process and code quality to stray away from established best practices—accumulating technical (Li et al., 2015) and process (Martini et al., 2020) debt. This made the codebase less accessible to newcomers, and led to some concerns when WWF decided that it would be beneficial to attract more external contributors and build an open-source community around Forest Foresight. To tackle the technical and process debt, we initiated a collaborative project between the domain experts and a team from Eindhoven University of Technology (TU/e). The project ran for a bit more than nine weeks, aiming to improve Forest Foresight's code and process, to expand the domain experts' knowledge on software engineering practices, as well as to thereby facilitate contributions by external developers.

In this article, we share our experience report of conducting the project and describe our insights of improving the Forest Foresight R package to open it up for an actual open-source community. We report our experiences from this practical project to contribute:

✉ `jduijvenbode@wwf.nl` (J.v. Duijvenbode); `i.d.wolff@tue.nl` (I. Wolff); `y.dajsuren@tue.nl` (Y. Dajsuren); `j.kruger@tue.nl` (J. Krüger)
ORCID(s): `0009-0001-3722-5022` (M.v. Apeldoorn); `0009-0003-8922-8780` (I. Wolff); `0000-0002-0283-248X` (J. Krüger)

[1] `https://forestforesight.atlassian.net/wiki/spaces/EWS/pages/33136/Overview`

[2] `https://www.icij.org/investigations/deforestation-inc/`

[3] `https://github.com/ForestForesight/ForestForesight`

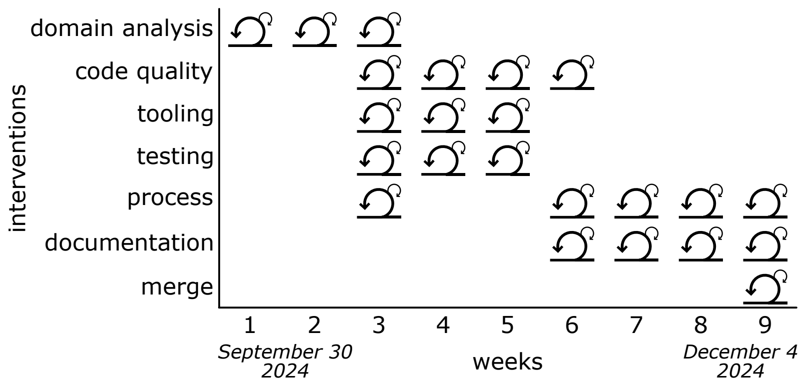

Figure 1: Timeline of our project, structured along the interventions we planned (y axis) and the weeks in which we executed them (x axis, Scrum Sprints along the action-research cycle). We detail how an individual Sprint looked like in Figure 2.

- Descriptions and discussions of changes we introduced to address technical and process debt. Specifically, we worked on reducing linter warnings, introducing continuous integration, implementing unit tests, defining review and release processes, adding GitHub issue and PR templates, as well as creating extensive documentation.
- Insights into how we communicated and trained the domain experts in software engineering practices. Most importantly, we experienced a positive impact of interactive training sessions on communicating and teaching best practices to domain experts.
- Impacts of R-specific challenges we faced when we introduced and adjusted software engineering practices. In detail, we experienced that the R language's design prohibits certain organizations of files that would be more accessible for large projects.

We hope that our contributions help software engineers interact with and train domain experts, providing guidance into how to incorporate software engineering practices into running projects and preparing these for community contributions. Moreover, our insights can help researchers working on software engineering for R, in training and education, or in research software engineering.

The remainder of this article is structured as follows. In Section 2, we report the conduct of our project, which represented a mixture of Scrum and an action-research-like process. Within Section 3, we provide a detailed description of Forest Foresight and our initial domain analysis. Afterwards, we detail the individual interventions we performed in Section 4 and discuss these in detail within Section 5. In Section 6, we suggest future steps to move Forest Foresight further ahead. Then, we discuss threats to the validity of our experience report in Section 7 and summarize related work in Section 8. Finally, we conclude this article in Section 9.

2. Project Conduct and Research Method

As we show in Figure 1, our project ran for a little more than nine weeks, from September 30, 2024, until December 4, 2024. Logically, the project was highly practice-oriented and we intermixed two processes to structure the daily workflow while also ensuring we could obtain valuable insights. We show how these two processes connected within an individual Sprint and who was primarily involved in each step in Figure 2. First, from a research perspective, we applied an action-research-like process (Staron, 2020), including iterative and collaborative interventions with evaluations of their impact. However, due to the project's practical and short nature, we typically executed individual cycles (i.e., interventions) in parallel.

Second, from an organizational perspective, we applied a Scrum process, with each intervention representing a larger backlog item. During each Scrum Sprint of one week, the core developers from TU/e (first three authors) performed research, planned their actions, and executed as well as evaluated these actions in collaboration with the domain experts—essentially refining the product backlog to define and execute a feasible Sprint backlog. During a Sprint, the TU/e core developers and domain experts reviewed each other's contributions, and code suggestions made by the TU/e core developers were used as a starting point and improved upon by the domain experts.

The Tu/e core developers also performed release and Scrum plannings, daily Scrums, as well as Scrum retrospectives. To guide their research and design decisions, they met once per week with their scientific advisors (last three authors) and at least once per week with the domain experts (not counting additional workshops and training sessions). These regular weekly meetings served as Sprint reviews (i.e., a scientific and a practical review), each taking around one hour. The scientific review involved reflecting on learnings and was also an opportunity to resolve any problems identified during the Sprint retrospective. Moreover, the TU/e scientific advisors helped with diagnosing problems to guide the release and Sprint planning. At the end of the project, the core developers gave a final presentation on their project, merged their contributions into the main branch of Forest Foresight, and submitted additional documentation they created on the project itself.

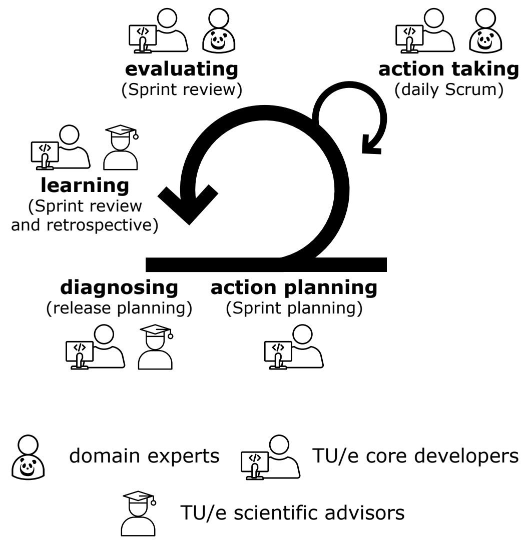

Figure 2: Mapping of action-research steps (in bold) to Scrum activities (in parentheses) within the individual Sprints of our project (cf. Figure 1), including the primary partners involved.

In Figure 1, we display the relations between high-level interventions (i.e., summarizing related smaller interventions) and Scrum Sprints. During the first two Sprints, we performed a detailed domain analysis of Forest Foresight, which essentially represents a larger *diagnosing* step within action research and resulted in a product backlog. All subsequent Sprints followed roughly the same procedure: The core developers started with a focused *diagnosing* step. They planned possible smaller interventions (*action planning*) and discussed these plans with the academic advisors during the weekly scientific Sprint review regarding feasibility, scientific resources, and timeline. Afterwards, the core developers updated their next Sprint backlog and then executed that Sprint (*action taking*). The *evaluating* step was challenging because we could not elicit concrete metrics on the benefits of, for example, code readability or process improvements. Instead, we relied on introducing and adjusting best practices from software engineering, gathering the domain experts' feedback on whether these were helpful to them. We derived *learnings* from the project during the scientific Sprint reviews through discussions between the core developers and academic advisors. During these discussions, we focused on reflecting on and analyzing R-specific adjustments, design decisions, as well as the training activities that we implemented to derive valuable lessons learned.

3. Site Description and Domain Analysis

In the following, we describe the Forest Foresight package, its initial status, the partners involved, and the goals of our project. Essentially, we summarize the results of our initial domain analysis (cf. Figure 1) to describe the site at which our project took place. Our domain analysis itself started with coordination meetings among the authors in which we agreed on the general directions of the project and everyone's roles. Within the first two Sprints, we refined these directions into concrete goals and possible interventions, reaching a principle agreement at the beginning of the third Sprint. To inform our discussions, the first three authors interviewed the domain experts, analyzed the Forest Foresight repository, and read up on relevant software engineering practices (explained in Section 4 for each intervention).

3.1. Forest Foresight

Originally started as "Early Warning System," Forest Foresight is an open-source R package hosted on GitHub.[3] The package's first release on GitHub was version 2.0.0 (September 2, 2024) under the GNU GPL-3.0 license, only a few weeks before our project started. On April 25, 2025, its most recent Version (4.3.1) has been released. Initially, Forest Foresight was developed by WWF in collaboration with the Boston Consulting Group (BCG), Deloitte, Amazon Web Services (AWS), and several academic institutions (WWF, 2022). However, please note that all code contributions to Forest Foresight until our project started came from the domain experts only.

Forest Foresight is a predictive early warning system designed to prevent illegal deforestation by forecasting potential deforestation sites. We display a screenshot of the Forest Foresight dashboard when it is running for Bolivia in Figure 3. The package is used in Suriname, Gabon, Sarawak (province of Malaysia), and Kalimantan (province of Indonesia), already averting illegal deforestation in Gabon and Kalimantan. Due to the success of the package, WWF plans to expand its use around these areas to cover the Amazon River, Guiana Shield, Congo basin, Borneo, and Sumatra. In addition, future rollouts of Forest Foresight are planned, for instance, in the areas of Gran Chaco (Bolivia, Paraguay, Argentina, Brazil), Cerrado (Brazil), eastern Africa, the Mekong River (China, Myanmar, Laos, Thailand, Cambodia, Vietnam), and the island of New Guinea (Papua New Guinea, Indonesia).

Forest Foresight uses near-real-time satellite data of these areas, including optical and radar imagery, as well as contextual information, such as, forest heights, proximity to oil palm mills, roads, elevations, slopes, and agricultural potential. It can incorporate user-provided data or preprocessed datasets that are available online, currently with a focus on the pan-tropical belt. Such data is fed into an XGBoost (Chen and Guestrin, 2016) machine-learning model that is trained on historical data with a six-month gap to identify regions at risk of deforestation in the coming six months. The model's predictions are grouped by geographical proximity and presented as prioritized alerts in a dashboard.

Given that Forest Foresight sometimes needs to operate on cost-effective hardware with limited RAM, R was selected as the programming language of choice due to its lazy loading and memory swapping. Other relevant design goals included that the code, specifically the package's main function `ff_run`, offers sufficient flexibility to handle a broad range of use cases. This shall enable users to tailor the machine-learning model to their needs without diverging from the Forest Foresight codebase.

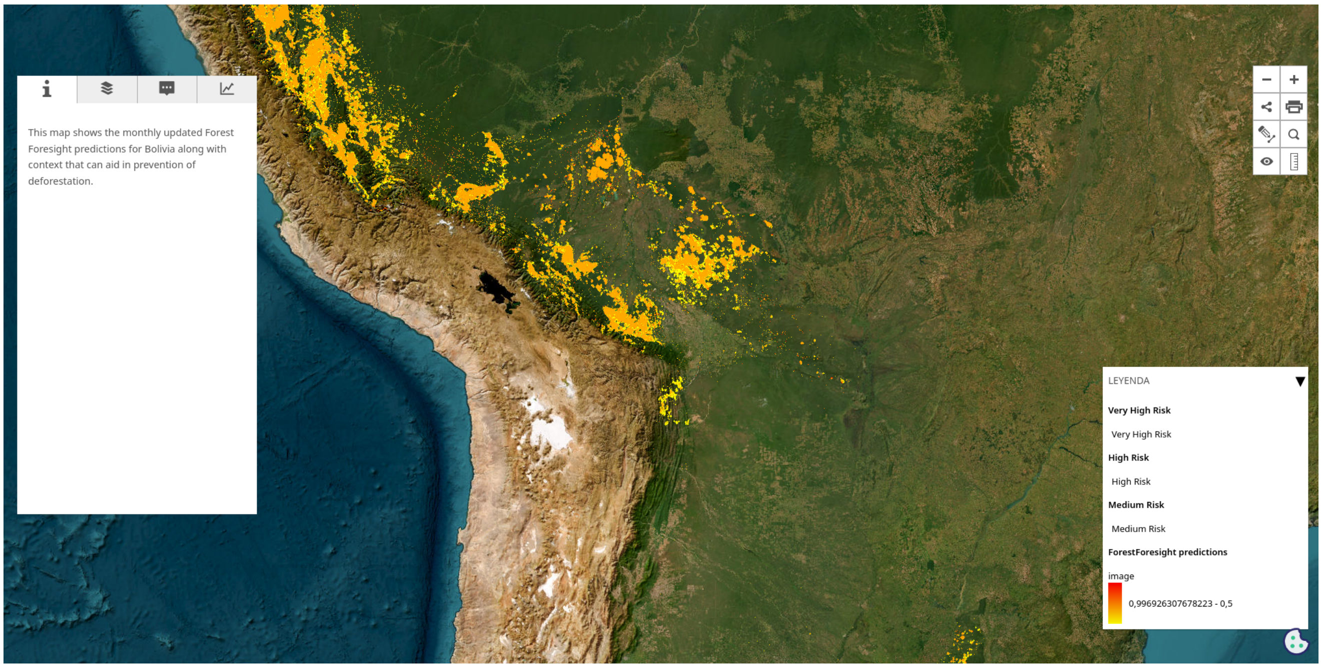
This map shows the monthly updated Forest Foresight predictions for Bolivia along with context that can aid in prevention of deforestation.
LEYENDA
Very High Risk
Very High Risk
High Risk
High Risk
Medium Risk
Medium Risk
ForestForesight predictions
image
0,996926307678223 - 0,5

Figure 3: A screenshot of the Forest Foresight dashboard for Bolivia.

Forest Foresight supports different spatial scales, leveraging 10x10-degree tiles for country-level or aggregated-tile analyses. The package achieves a precision of up to 80 % and allows local actors to prevent illegal deforestation (WWF, 2022)—which, otherwise, is typically identified only after damage has been done.[1] So, the key idea of Forest Foresight is to intervene **before** illegal deforestation takes place, taking preventive measures against environmental destruction.

3.2. Roles of the Partners

WWF Domain Experts. WWF was founded in 1961 as a Swiss-based international non-government organization to protect the environment, focusing on preserving wilderness, reducing human impact, avoiding natural loss, and fighting climate change. Today, WWF has more than five million supporters worldwide and has invested over $ 1 billion into over 12,000 projects. As one of such projects, Forest Foresight is managed by two core developers with whom we collaborated to decide on and execute our interventions. Consequently, the fourth and fifth (product owner of Forest Foresight) authors of this paper served as contact persons for the core developers. So, the domain experts provided input on their needs and finally approved what interventions to explore, implement, as well as keep.

TU/e Core Developers. From TU/e side, there were three core developers (first three authors). They executed this project as part of their EngD programs as an in-house project. The EngD program is a two-year post-master program involving four technical universities in the Netherlands in which graduate students participate in courses and practical in-house projects.[4] In the end, EngD trainees conduct a ten-month final design project typically in collaboration with an industrial partner. All three core developers were part of the Scrum team and involved in all tasks of the project. So, they were responsible for communicating with the other partners, proposing and implementing interventions, as well as reporting on their results. The first author acted as project manager, the second as designer, and the third as Product Owner as well as Scrum Master of the team.

[4]https://www.4tu.nl/sai/

TU/e Scientific Advisors. To guide the core developers in their project, there was a group of three advisors from TU/e (last three authors). The advisors were responsible for setting up the project, meeting with the core developers to review and plan their progress (scientific Sprint reviews), and suggesting relevant research materials. As such, they helped the core developers explore relevant software engineering practices and publications to scope their interventions. Furthermore, the advisors supported the fine-tuning of interventions and judging their feasibility within the given time period. During the regular meetings, the advisors also reflected with the core developers on the learnings we report in this article.

3.3. Forest Foresight at the Start of the Project

When our project started, Forest Foresight was operational and employed in four areas around the world (cf. Section 3.1), but still within its pilot phase. The domain experts had no formal software engineering education, and thus were not familiar with best practices on development processes, design principles, or other relevant practices and research. They felt that the codebase was not accessible to newcomers, which harmed external contributions. Through our domain analysis, inspection of the codebase, and discussions with the domain experts, we identified several code and process issues, which served as input for scoping the project according to the domain experts' needs.

Code Issues (CIs). When we analyzed the code of Forest Foresight, we identified several areas for improvement:

CI_1 Using the `lintr` library from CRAN,[5] we received 1,795 warnings for the entire codebase. The majority of these warnings involved styling issues of the code, which was not consistent with any commonly used styleguide. Other warnings were related to, for instance, visibility modifiers, line lengths, or symbols that should be actual Boolean values.

CI_2 We found that the median length of a function was 49 lines of code, and some functions had several hundreds.

CI_3 We noticed that no unit tests were implemented.

CI_4 When inspecting `ff_prep` as the package's interface, we personally considered variable names and 15 function names as "vague;" with the domain experts agreeing with this perception. Essentially, their purpose was hard for us to understand without reading the surrounding code, obtaining additional domain knowledge, or asking the domain experts.

CI_5 Relating to CI_4, we found that the documentation of the code and its development process was outdated.

CI_6 We also found that configuration parameters were hard-coded variables in the source code. Consequently, if a user wanted to adjust Forest Foresight to their needs, they needed to modify the actual source code.

CI_7 We identified that dependency management within and beyond the package was not standardized, and thus could easily cause breaking changes and reproducibility issues for users relying on different environments.

Such code issues are well-established indicators of quality problems in the design and implementation of software, typically complicating program comprehension and maintenance (Martin, 2008; Fowler, 1999; Zeller, 2009; Krüger and Hebig, 2023; Li et al., 2015). We felt that these were clear directions for improving the code so that it would be easier and more motivating for external developers to contribute to Forest Foresight.

Process Issues (PIs). By inspecting the GitHub repository and discussing with the domain experts, we aimed to understand their development process, which revealed some complementary areas for improvement:

PI_1 We learned that external parties that wanted to use or adjust Forest Foresight sent feature requests and bug reports via e-mail. Consequently, discussions revolved through different mail clients in an unorganized fashion, and the submissions had to be developed and integrated by the core developers.

PI_2 Related to PI_1, we found that there was no explicitly defined development process that would cover, for instance, when to branch, fork, or merge.

PI_3 Due to CI_5 and PI_2, we were not surprised that there were no contribution guidelines, like a `contributing` file or templates for issues and pull requests.

PI_4 We noticed that Forest Foresight employed semantic versioning, but did not strictly follow it to label releases in a meaningful way. For instance, GitHub lists only two releases (2.0.0, 3.0.0) for the time prior to our project.

PI_5 Since unit tests were missing (CI_3) and no process was defined (PI_2), it was not surprising that automated continuous integration was missing.

Such process issues are the result of not adhering to typical software engineering practices. Due to the persistence of the issues, the domain experts and we noticed that their workload increased while external developers had trouble contributing directly to Forest Foresight—essentially accumulating process debt (Martini et al., 2020).

3.4. Goals of the Project

Domain Experts' Needs. When we first discussed the project with the domain experts, we identified three primary goals (G) they wanted to achieve for Forest Foresight:

G_1 as a short-term goal, the package's codebase should be improved to make it easier to understand and maintain;

G_2 as a medium-term goal, the entire project should be opened up towards attracting an open-source community around Forest Foresight; and

G_3 as a long-term goal, a vision on how to maintain and continue the project should be developed.

We discussed these goals among all partners and mapped them to the areas of improvement we identified through our domain analysis (cf. Section 3.3). Throughout the first two Sprints, we refined this mapping and planned possible high-level interventions. We discussed these interventions with the domain experts and specified high-level requirements to fix our agreement for the project.

In Table 1, we present an overview of the high-level requirements we agreed on. As we can see, most of the requirements connect to G_2, aiming to introduce a sustainable change that opens Forest Foresight for open-source contributors. The code improvements related to G_1 were substantial, but we considered them also as a prerequisite to achieve G_2. Thus, we agreed that we would start with improvements to the codebase but would also tackle the medium-term goal of the domain experts. For G_3, we agreed that we would propose some future steps to explore, but we considered them to exceed the scope and timeline of the project. Specifically, we did not plan concrete interventions for G_3, but agreed on exploring machine-learning algorithms

[5] `https://cran.r-project.org/package=lintr`
DOI: 10.32614/CRAN.package.lintr

Table 1
Overview of important high-level requirements agreed on by all partners. Approved means that our interventions had been integrated at the end of our project, while draft refers to proposals on which the domain experts still had to decide on.

requirement	issues	goal	final status
improve the codebase to make it more maintainable	CI_1, CI_2, CI_4, CI_5, CI_6	G_1	approved
implement a testing strategy	CI_3, PI_5	G_1, G_2	approved
improve and update the documentation for developers and users	CI_5	G_1, G_2	approved
simplify dependency management to avoid breaking changes and facilitate reproducibility	CI_7	G_1, G_2	approved
alert users not familiar with GitHub about new releases	CI_7, PI_4	G_2	draft
introduce and document open-source collaboration practices	PI_1, PI_2, PI_3, PI_4	G_2	approved
introduce and document issue and bug reporting	PI_1, PI_3	G_2	approved
introduce and document a development strategy	PI_2	G_2	approved
introduce and document a versioning principle	PI_4	G_2	approved
implement a continuous-integration pipeline to prevent breaking changes	PI_5	G_2	approved
investigate alternative machine-learning solutions	—	G_3	draft
propose future steps for improving the project	—	G_3	approved

and management as well as a proposal for future improvements of Forest Foresight; especially on its visibility for open-source developers.

Envisioned Interventions. As described, laying the foundation for a healthy open-source community around Forest Foresight manifested as the primary goal of our project (G_2). Through this, the domain experts hoped to support the package, engage more users, and increase the package's visibility. To achieve this goal, we aimed to address the issues we identified to reduce the barriers of onboarding and contributing to Forest Foresight. In detail, we planned five high-level interventions (cf. Figure 1), which we detail in Section 4: 1. code quality, 2. tooling, 3. testing, 4. process, and 5. documentation. Please note that we did not always follow the potentially "best" software engineering practices. This was a deliberate choice to not overload domain experts who also need to contribute to and who use Forest Foresight. Consequently, we made some adjustments to simplify a practice or make it more fitting to the domain experts' experiences and development style.

Training. An important part of the project was to train the domain experts on software engineering practices. For this reason, the core developers did not simply implement the changes and explain them, but followed an extensive training process. Specifically, for each intervention, they first implemented parts themselves to demonstrate by example. For instance, they refactored parts of the codebase, introduced a set of unit tests, set up a continuous-integration pipeline, proposed issue and pull request templates, and created pull requests. Then, they met with the domain experts to explain an applied principle, introduce new tooling, and show how it works, typically combined with workshops or reviewing sessions. In such workshop session, the domain experts would review the core developers' work, apply the practices themselves, and the core developers would then review the domain experts' work. Afterwards, they discussed the results and any uncertainties. This way, we tried to transfer knowledge and expertise based on applying the practices on the actual project. In the end, we had at least one of such workshop sessions of around two hours for each intervention. After one to two weeks after such a session, there was another one to two hour session to discuss any problems the domain experts experienced and to identify how to resolve these. The final say on accepting an intervention and integrating the proposed changes into Forest Foresight remained with the domain experts. To track their progress and create reusable documentation, the core developers documented their work and the final decisions agreed on with the domain experts.

4. Interventions in Detail

In this section, we report on our five core interventions in detail (from second to second-to-last in Figure 1). For this purpose, we follow the steps of action research (Staron, 2020) to structure our experiences.

4.1. Code Quality

Diagnosing. To attract an open-source community, we aimed to make the code of Forest Foresight as comprehensive and structured as possible (G_1). This means that the codebase should be easy to read, organized, documented, and follow established naming conventions. During our domain analysis, we noticed that the code struggled along these dimensions, seeing, for example, the 1,795 linter warnings (CI_1), long methods (CI_2), and vague function/variable names (CI_4). When we analyzed the code in depth and iterated through multiple improvements, we noticed that such issues hampered our comprehension of the code.

Action Planning. To derive concrete improvements on code quality, we built on our experiences, inspections of other R packages, and research on, for example, software-design principles (Robillard, 2019), refactoring (Fowler, 1999), clean code (Martin, 2008), and identifier naming (Binkley et al., 2013; Beniamini et al., 2017). Based on this, we iteratively introduced the following interventions:

Cleaning Up Code: We started with cleaning up the code, meaning that we employed refactorings to resolve linter warnings (CI_1), to split up long methods (CI_2), and to clarify vague identifier names (CI_4). For this purpose, we also intended to introduce an established code-style standard that should be adhered to.

Adding Code Documentation: In parallel to our other interventions, we planned to add documentation (i.e., comments) within the code to provide a natural-language explanation of complex code excerpts (CI_5).

Introducing Single-Responsibility Principle: After splitting long methods, we proposed to further reduce the code complexity by adhering to the single-responsibility principle by leveraging functional decomposition (CI_2). For this purpose, we planned to restructure the functions into cohesive units of functionality and to introduce a folder structure within the codebase.

Extracting Configuration Options: During our previous Sprints on this intervention, we noticed and documented hard-coded configuration options in Forest Foresight. During the later Sprints, we decided to extract these into a dedicated configuration file to allow users to configure the library without having to modify the code itself (CI_6).

Through these smaller interventions, we aimed to improve the quality and understandability of the codebase.

Action Taking. At first, the core developers implemented the interventions in a separate fork, refactoring parts of the codebase. Then, they reviewed their changes with the domain experts during training workshops to identify whether the refactored code was more comprehensive to them, too. Upon agreeing on the extent of refactoring, the core developers and domain experts continued to independently improve the code. Afterwards, they reviewed each others' changes to reflect on which improvements were perceived beneficial by all of them. In the end, the entire code was refactored, and finally approved by the domain experts.

To judge whether a piece of code needed to be refactored, the core developers defined a set of questions, such as:

- Is the code easy to understand?
- Can complexity be reduced by refactoring the code?
- Is the code well organized?
- Are identifier names intuitive and obvious?
- Is the code well documented?
- Does the code meet the code-style standard?

Each developer's understanding and the introduced code-style standard served as basis to answer these questions. As standard, the core developers proposed to employ the Tidyverse styleguide.[6] Based on our research, this standard is widely used in the R community, meaning that it is likely for relevant contributors to be familiar with the styling of the codebase. Also, this standard has the advantage of being compatible with the `lintr` library to verify it, while the CRAN library `styler`[7] supports some automatic formatting.

[6]https://style.tidyverse.org/

Evaluation. The domain experts approved and merged the suggested code improvements into the `main` branch of Forest Foresight. Their feedback was very positive and the improvements were appreciated. In terms of concrete metrics, we reduced the number of linter warnings from 1,795 down to 102, the median lines of code per function from 49 to 21, and fixed 14 of the 15 vague function names we identified before. In general, the positive feedback and the considerable changes in the metrics indicate that our interventions were valuable contributions to improving Forest Foresight.

Learning. Introducing basic software engineering principles on code style and formatting already had a positive impact. Consequently, communicating the benefits of these practices became simple in our case. Essentially, the core developers initially had the role of newcomers onboarding to the project: They had software engineering knowledge, but no expertise in the domain or with the project. Their subsequent confusions and clarification questions were clear indicators for code improvements to facilitate the onboarding of other external developers. Moreover, the domain experts rapidly caught on to these practices, since they experienced their benefits first-hand when working on the code themselves. So, the issue was not about learning these practices, but simply the missing background knowledge, and thus a lack of awareness.

Regarding R, we experienced that breaking down complex functions into smaller units was restricted by limitations of the language and R-package specifications. When we conducted our project, R packages did not allow developers to use subdirectories within the `/R` folder, restricting code organization to different files in a single directory (Wickham and Bryan, 2023). This limitation prevented us from creating a hierarchical folder structure to fully support the single-responsibility principle, which could have further improved the code. Other solutions exist, for example, the package `box` on CRAN,[8] but we considered these to be out of scope of this project. Particularly, we were reluctant to introduce completely new and complex concepts to enforce another practice. Instead, we proposed to break down complex functions into smaller units either in the same file or in separate files in the same directory.

4.2. Tooling

Diagnosing. Forest Foresight initially lacked tools for effective collaborative development. Dependency management was not straightforward (CI_7), there were no tools for style checking (CI_1), and a continuous-integration pipeline was missing (PI_5). Introducing the respective tooling reduces manual development effort and provides immediate feedback to external developers on their contributions (G_1, G_2).

[7]https://cran.r-project.org/package=styler
DOI: 10.32614/CRAN.package.styler

[8]https://cran.r-project.org/package=box
DOI: 10.32614/CRAN.package.box

Please note that we introduced the tooling in parallel to other interventions, since these are interdependent.

Action Planning. To scope interventions, we inspected GitHub's continuous-integration guidelines,[9] and how other R projects check their code and manage dependencies. This led to three plans:

Systematizing Dependency Management: To ensure a consistent, reproducible environment for all contributors and users, we proposed standardizing the dependency management using the package `renv`[10] (CI_7).

Implementing Continuous-Integration: We proposed to implement a continuous-integration pipeline using GitHub Actions to automate various checks, such as build success, unit testing, style compliance, and R style validations (PI_5).

Introducing Style Checker: As we describe in Section 4.1, we reformatted the code to adhere to the Tidyverse styleguide. Logically, we also integrated the respective packages `lintr` and `styler` into code reviews and the continuous-integration pipeline (CI_1). To not reiterate the same content, we do not discuss the specifics of the style checker separately below.

Through these interventions, we aimed to reduce the developers' effort, maintain the quality of the codebase, and implement automated feedback for external contributors.

Action Taking. The core developers introduced the R package `renv`, which allows users to create reproducible, isolated, and portable environments by setting up a private library for each project. For this purpose, they followed the setup instructions to integrate `renv` into Forest Foresight and to snapshot the dependency versions used. Now, `renv` is used automatically when the project is opened and maintains a lock file with the package versions to ensure consistent dependencies. Furthermore, the core developers relocated development-only packages to the description file's `Suggests` field to mark them as non-essential.

To automate repetitive tasks, the core developers implemented continuous integration pipelines. Since Forest Foresight is hosted on GitHub, they used the well-established GitHub Actions, which provides all necessary features and is free for open-source projects. Specifically, they implemented the following pipelines:

Build and Release (`build_release.yml`) builds and releases the Forest Foresight package if a pull request is merged into the `main` branch.

Test and Check (`r_test_and_check.yml`) executes all unit tests and the `R CMD check`[11] on any code changes that target the `main` or `develop` branches.

`Styler` and `Lintr` (`styler_lintr.yml`) formats the code using `styler` and lints the code with `lintr` on any code changes that target the `develop` branch.

If any of these pipelines fails on a pull request, the creator of that pull request will be notified, and the merge will be blocked until all failing checks have been addressed.

Evaluation. The core developers introduced the new tooling to the domain experts during training sessions, explaining how they work and when they are triggered. Although we cannot compute concrete metrics on the impact of the tools, they have been used since. Overall, the domain experts' perception about the tools was positive again, and they have helped maintain the quality of the code since.

Learning. The tools we introduced are well-established means to maintain code quality and automate checks on code contributions. It is not surprising that they benefited Forest Foresight, too. Similarly to the development practices we introduced, one reason that dependency management and continuous integration had not been used before was the lack of awareness for such tools. However, setting up these tools was also more complex than introducing development practices, which required us to read technical documentation in detail. This step seems like a substantial, logical barrier that requires more in-depth software engineering and technical background than we should expect from domain experts.

4.3. Testing

Diagnosing. To find bugs, ensure robustness, check code quality (Section 4.1), and enable continuous integration (Section 4.2), testing is essential. However, at the beginning of the project, Forest Foresight lacked any unit tests (CI_3) or further test automation (PI_5). Consequently, quality assuring the package was challenging and primarily a manual effort, also limiting the possibility to test contributions by external developers immediately.

Action Planning. Our primary action for this intervention was straightforward to define: introducing automated unit testing to quality assure code and changes (CI_3, PI_5). Unit testing is a well-established strategy to test the correctness of individual functions in isolation under predefined conditions. Through automation, unit tests can be executed on code changes to prevent them from introducing unintended behavior (Mårtensson et al., 2019). We built on our knowledge and established practices (Runeson, 2006; Daka and Fraser, 2014) on unit testing to set up the test environment and test cases for Forest Foresight.

By inspecting R resources and other packages, the core developers identified `testthat` (Wickham, 2011)[12] as the most widely used unit-testing framework for R. It offers a testing API and various tools to execute tests and evaluate their results. We proposed to introduce `testthat` for Forest Foresight and suggested to focus on two ways of defining unit tests to keep the effort manageable:

[9] https://github.com/resources/articles/devops/ci-cd

[10] https://cran.r-project.org/package=renv
DOI: 10.32614/CRAN.package.renv

[11] https://r-pkgs.org/R-CMD-check.html

[12] https://cran.r-project.org/package=testthat
DOI: 10.32614/CRAN.package.testthat

Testing Expected Behavior: These tests evaluate whether a function behaves as expected. They do not cover all possible execution paths or edge cases (i.e., robustness) unless the developers consider these important.

Defining Tests on Regression: When a bug or issue with the codebase is reported, a test case is created that reproduces the problem. Once the problem has been resolved and the test passes, the test ensures that the issue remains resolved. We planned to iteratively use these tests to demonstrate how to move towards regression testing.

Please note that we did not focus on robustness tests during our project to reduce complexity and effort. The current testing strategy ensures that Forest Foresight performs as expected for the main use cases and prevents regressions over time. In parallel, the testing does not overwhelm domain experts with new technologies and complexity.

Action Taking. To get started with `testthat`, the core developers set up the `testthat` infrastructure and a simple unit test. After familiarizing themselves with the framework, they developed more unit tests and created mock data. In parallel to introducing the continuous-integration pipelines (cf. Section 4.2), they conducted training sessions to explain how the tooling and unit testing worked to the domain experts. The pipelines also introduced regression testing. Throughout the project, the core developers and domain experts added unit tests, implementing 103 at the end of the project. Of those unit tests, 65 were implemented by the domain experts.

Evaluations. After our project, Forest Foresight had 103 unit tests, instead of zero. The tests are used to check pull requests and have proven useful, which is why the domain experts continue to use and expand the test suite. In fact, as we described in Section 4.2, the tests are executed for any changes introduced to the `main` or `develop` branches. Even though the test suite is not complete (e.g., missing robustness tests), and thus is also not perfect for regression testing, the domain experts were convinced by the testing strategy. This is evidenced by the 65 test cases they implemented themselves, and their commitment to sustain the test coverage.

Learning. Introducing a testing strategy can be an overwhelming task, due to the many options and tools available. In our experience, the `testthat` framework works well for R and can be easily trained to domain experts. To reduce workload and complexity, we deliberately decided to first focus on the most-established testing strategy of unit testing and a subset of all relevant testing purposes. We did also not dive into coverage criteria or other more in-depth testing topics. Our reduced testing strategy felt like a feasible balance between avoiding overly complex testing while covering the most important error sources. By laying these foundations, we enabled the domain experts to expand the existing test suite and to expand the testing strategy according to their future needs.

4.4. Process

Diagnosing. To motivate external developers to contribute and to manage community contributions, defining processes like reporting a bug, proposing a feature, or submitting a contribution should be defined. However, while hosted on GitHub, Forest Foresight did not make use of GitHub's well-established support for community collaboration and processes (PI_2, PI_3). Instead, the domain experts communicated primarily via e-mail (PI_1) and through a Discord. Lastly, while semantic versioning was known, it was not enforced as part of the release process (PI_4). In summary, at the beginning of the project, the core developers felt that it was largely unclear for externals or onboarding developers how Forest Foresight was developed and what communication channels to use—which was also related to the outdated documentation (cf. Section 4.5).

Action Planning. As an initial step to investigate process improvements, we studied research on barriers for newcomers who want to join open-source projects (Steinmacher et al., 2015, 2019) and on contribution guidelines (Fronchetti et al., 2023; Krüger et al., 2020). At this point, it became apparent that defining a feasible open-source development process is a complex topic that exceeded the scope of our project. Consequently, we decided to narrow down our scope to adapting a smaller set of well-established practices.

For this purpose, the core developers inspected the development processes used by two successful open-source projects. First, the Linux kernel[13] is one of the most prominent and largest software systems ever developed, and is still organized via mailing lists (Schneider et al., 2016). Second, the Marlin 3D printer firmware[14] follows a more typical community-driven process, while still being coordinated by core developers (Krüger et al., 2019). We selected these two systems, because they have substantial communities, a form of central authority (i.e., resembling the domain experts for Forest Foresight), and contrast the two development strategies (i.e., e-mail versus GitHub). As a consequence, we considered these two systems as feasible subjects to showcase how the domain experts could keep the mailing system (i.e., Linux process) or move more towards GitHub (i.e., Marlin process).

Based on our analysis, we proposed to the domain experts to use GitHub instead of e-mail to take advantage of:

- A simple and integrated issue-tracking system that is less intimidating for newcomers compared to a mailing list, and which is well-known to many open-source developers.
- Collocating issues, pull requests, and developer discussions with the code repository, establishing a central communication platform managed by the experts.
- Facilitated organization and discovery of issues and pull requests via labels and filters.

[13]https://www.kernel.org/
[14]https://marlinfw.org/

We proposed these changes because Forest Foresight is currently a smaller open-source project that aims to attract a community. Thus, simplicity and openness are key, while developing Forest Foresight does currently also not require more elaborate structures. For these reasons, we considered the move towards a single platform for all relevant artifacts and communications to reduce organizational overhead and to lower newcomer barriers.

To move towards GitHub, we planned four interventions:

Introducing Templates: GitHub allows developers to define issue and pull-request templates to ask for specific information. We proposed to introduce such templates to make it easier for external developers to interact and to understand what pieces information should be provided, also reducing the need for back and forth discussions (PI_3).

Managing Issues and Pull Requests: We aimed to define and document a workflow for maintainers for moderating issues and pull requests (PI_1, PI_3). This way, we wanted to ensure that important properties are checked, that contributors receive feedback, and that maintainers act consistently.

Enforcing Code Reviews: To maintain code standards, we suggested mandatory code reviews for any new change to the `develop` and `main` branches (PI_2).

Defining Workflows: To remedy the absence of a branching strategy, we proposed implementing a simplified version of Gitflow[15] as a starting point (PI_2).

Automating Semantic Versioning: To improve the adherence to semantic versioning (PI_4), we discussed its details with the core developers and implemented automated releases, as we described in Section 4.2.

Our goal with these interventions was to make Forest Foresight easier to engage with by defining consistent processes for submitting, reviewing, and integrating contributions.

Action Taking. The core developers introduced templates for issues and pull requests following GitHub's documentation.[16] Based on their research on other R packages and Marlin, they developed three templates:

1. Bug reports (issues) to allow others to report misbehavior of Forest Foresight using fields to describe *the bug*, steps *to reproduce it*, the *expected behavior*, *screenshots* if applicable, and *additional context* for any other details.

2. Feature requests (issues) to allow others to propose new features for Forest Foresight using fields to describe *the specific problem to solve*, *the envisioned solution*, *potential alternatives*, and *additional context*.

[15]https://nvie.com/posts/a-successful-git-branching-model/

[16]https://docs.github.com/en/communities/using-templates-to-encourage-useful-issues-and-pull-requests

3. Pull requests allow developers to submit their contributions for integration into Forest Foresight, for which they are asked to describe *the pull request*, *requirements*, relevant *configurations*, and *related issues*.

The core developers implemented and tested these templates iteratively, conducting training sessions with the domain experts to show them how these templates work and to collect their feedback.

Regarding workflows, the core developers tackled two aspects. First, they defined a workflow for maintainers on how to moderate issues and pull requests. The proposed workflow specifies when and how to close duplicated or out-of-scope issues and how to ensure that pull requests meet the project's new quality standards via code reviews and continuous integration. As for most open-source systems that are developed on GitHub, external developers can work on issues in a personal fork and propose their changes via a pull requests, which must be approved and integrated by a maintainer of Forest Foresight.

Second, for the internal workflow by maintainers, the core developers introduced Gitflow as a comprehensive means to coordinate branches. To ease the adoption, they introduced basic concepts like feature branches and how these interact with the `main` and `develop` branches. In parallel, they allowed more flexibility to not introduce any unnecessary constraints that are hard to explain—and that are not necessarily a best practice for every software project.[15] For instance, through training sessions with the domain experts, the core developers decided to omit concepts like `hotfix` or `release` branches. Instead, they aimed to align more with trunk-based development, which has become more popular.

A substantial change in the domain experts' workflows was the introduction of systematic code reviews. To guide maintainers, the core developers proposed to use the same questions as for code quality (cf. Section 4.1) as reference. In addition to these questions, the maintainers should also inspect whether the new or changed code is tested to support future continuous integration (cf. Section 4.4). By using these questions, we hoped to motivate maintainers to critically reflect on a pull request and ask questions for clarifications or changes before approving a pull request.

Evaluating. We introduced and adjusted our interventions through training sessions, in which we received positive feedback from the domain experts. They quickly familiarized themselves with the process changes and applied them for their own development of Forest Foresight. In fact, the project had no issues or pull requests when we started the project. By the end of the project, we identified 11 open and 23 closed issues, two open and 31 closed pull requests (24 successful merges), and 256 executions of the continuous-integration pipelines. There was only a single direct push to the `develop` branch. In all other cases, the domain experts and core developers followed the introduced Gitflow. We remark that the core developers introduced only a small set of issues to enable the training. Already during the remainder of the project, the domain experts maintained the issues and pull requests. The domain experts have continued to actively make use of the new processes. As of July 1, 2025, Forest Foresight has seven open and 50 closed issues (23 more) as well as two open and 72 closed (63 merged) pull requests (39 more). These numbers and the visible shift in the development practices underpin that the introduced interventions are perceived helpful by the domain experts and can likely ease contributions by external developers.

Learning. Software-development processes can be complex and challenging to grasp for domain experts without the respective background. To mitigate these problems, we simplified many process steps and practices, aiming to introduce fundamentals during our project. We found such simplifications very helpful for three reasons. First, it became easier to introduce software engineering practices to the domain experts. Second, it allowed us to focus on some fundamentals and show their benefits for developing Forest Foresight and potentially attracting open-source contributors. Third, starting with a few fundamentals made it simpler to scope and customize these to the domain experts' needs. Seeing the improvements and guidance of the implemented process changes quickly convinced the domain experts of the interventions' benefits.

4.5. Documentation

Diagnosing. At the beginning of the project, the core developers were challenged by a lack of up-to-date documentation of Forest Foresight, for code and processes alike (CI_5). Inadequate or unclear documentation can significantly hinder developers in attempting to onboard and contribute to an open source project (Steinmacher et al., 2019), while a comprehensive `contributing` file can be highly beneficial (Fronchetti et al., 2023). Unfortunately, documentation is often poorly maintained, even though it can be helpful to, and is appreciated, by developers (Krüger and Hebig, 2023; Nielebock et al., 2019). Consequently, throughout the initial Sprints, it also became apparent that maintaining the documentation of Forest Foresight would have helped tremendously with onboarding the core developers and recovering knowledge. Similarly, version releases did not clearly document the nature of the introduced changes, incrementing only major releases and containing no information, for instance, on breaking changes.

Action Planning. We planned two types of improvements:

Commenting Source Code: The benefits of commenting source code are often debated in software engineering (Nielebock et al., 2019; Martin, 2008). Still, we proposed to comment complex pieces of code to help communicate and clarify the intentions of code among domain experts and open-source developers (CI_5).

Introducing Contributing Guidelines: As an extensive effort, we agreed to document the process changes we introduced (cf. Section 4.4), propose a respective `contributing` file to help onboarding developers, and update the `readme` of Forest Foresight (CI_5, PI_2, PI_3).

By creating the respective documentation, we intended to facilitate contributions from (external) developers by providing comprehensive explanations of the project and its development processes.

Action Taking. While improving the codebase (cf. Section 4.1), the core developers added comments where they considered them feasible and reviewed these with the domain experts. Furthermore, they created notes and presentations on the project to record their progress. While introducing the interventions, they also created respective documentation, for instance, on coding-style standards, continuous integration, workflows, issue and pull-request templates, or the source code. They synthesized their notes into the `contributing` file, templates, and internal reports. Lastly, they reflected on their insights from researching Forest Foresight and mapped these to the `readme`, proposing fixes to update the information contained in it.

Evaluating. At the end of the project, the core developers had added extensive documentation to Forest Foresight, most importantly the `contributing` file as a central resource for external developers. We consider this a substantial upgrade to Forest Foresight, since we personally experienced the challenge of familiarizing with the project when lacking necessary domain knowledge. The domain experts agreed on the substantial updates and improvements the core developers implemented.

Learning. While developers have mixed opinions about code documentation, we found it tremendously beneficial to record domain specifics. This includes comments in the code to explain complex or domain-specific constructs explicitly at the respective locations as well as process documentation. As underpinned by previous research, we perceived particularly the `contributing` file as helpful, also for internal use.

5. Lessons Learned

Interventions and Forest Foresight. Forest Foresight is an important R package to prevent deforestation. We aimed to improve the package and its implementation, helping the domain experts to hopefully attract an open-source community in the future. Compared to the beginning, all parties of the project agreed that our interventions improved the package. For example, we reduced linter warnings (from 1,795 to 102), introduced continuous integration, implemented 103 unit tests, defined processes, added templates, and created extensive documentation. Most of these things were missing at the start of our project, but are fundamental to improve any software system and its development. During the project, it became clear that such established software engineering practices are highly valuable, and were only missing due to the domain experts not being aware of these.

Consequently, as we display in Table 1, almost all of our interventions were in the end approved by the domain experts to fulfill the defined requirements. Only two of the requirements were still in the status "draft." First, we did not introduce an automatic means for alerting users that are not familiar with GitHub about new releases, but suggested several options that the domain experts have to coordinate with their users. Second, we investigated improvements for the algorithms and for managing the machine-learning components of Forest Foresight, but introducing and benchmarking these exceeded the agreed scope of our project. Neither of these two requirements being in a draft status limits the success of the project, with all parties agreeing that these must be addressed in future projects and decided upon by the domain experts. Similarly, we identified and proposed a series of future improvements on software engineering practices, which we report in Section 6.

In essence, we can summarize our key insights into our interventions regarding our case as follows:

Interventions on Software Engineering Practices

• Established software engineering practices had a positive impact. • Improving the code structure helped both the joining and the original (domain experts) developers. • While using established tools was very helpful, some more advanced ones (e.g., continuous-integration pipelines) were complex for the domain experts and even the core developers to set up. • Defining and documenting templates, guidelines, and recommendations helped the domain experts stick to software engineering practices. •

R Programming Language. Forest Foresight started as an R package because this allowed to run it on cost-effective hardware. Moreover, the language involves features that simplify programming, many packages, and was known to the domain experts. Such properties made it a feasible choice. Still, with the project growing and becoming more complex, we experienced that using R became more challenging.

Most importantly, as we described in Section 4.1, the R language and its package specifications imposed limitations on code organization. Specifically, the specifications prohibited developers from organizing their R code in subdirectories in the `/R` folder. The official documentation on this limitation, the underlying design decisions, and the recommended solutions were sparse. Regardless of the intentions behind this decision, it can be problematic for developers of a package that grows beyond a certain size. A flat hierarchy requires a developer to either accept suboptimal code organization, or to split the package into multiple packages. However, splitting a package may not fit the conceptual boundaries of the package. Also, resorting to third-party solutions may run the risk of falling out of sync with the core language specification. For our project, this situation made it more complicated to enhance the codebase.

While R experts may have a good solution for this flat hierarchy, domain experts without this background and limited time will likely not be able to identify or implement such a solution. Consequently, projects driven by domain experts, such as Forest Foresight, should carefully consider what programming language to use and what software engineering practices work with that language. In the end, the availability of relevant out-of-the-box features and ease of use for domain experts are often the key factors for selecting a programming language in such projects. Thus, researchers should investigate how to identify and transfer best practices for languages that have been considered less often in the past.

In essence, we can summarize our key insights into R regarding our case as follows:

R for Software Projects

• R was a reasonable choice at the start of developing the package due to its accessibility to the domain experts. • R imposed constraints that made it challenging to implement certain software engineering practices and principles. • R offered various supporting packages with varying degrees of visibility and accessibility. •

Training. Our project involved extensive training activities for domain experts without software engineering background. In our experience, it worked best to start with basic software engineering practices, stripping them down to their essence and introducing them by example. Then, the domain experts can implement the demonstrated principles themselves in their project, allowing for collaborative review sessions to gather each others' feedback. Through this feedback, it is possible to identify confusions, scope adjustments and simplifications, or decide to deepen certain practices. These collaborative review sessions also helped us reflect on the trade-offs of the introduced practices within the projects' specific context and the responsible developers' work style. Lastly, we felt that documentation was particularly important for our project to establish a coherent reference for all involved and future parties, independently of their expertise.

In essence, we can summarize our key insights into training domain experts as follows:

Training Domain Exerts

• Simplifying software engineering practices made it easier to introduce these to the domain experts. • Starting with interventions that yield immediate improvements made it easy to demonstrate benefits. • Interactive training sessions with learning-by-doing were key. • Core causes for the domain experts not using software engineering practices before were a lack of awareness of these or misunderstandings of their intent. • Documenting practices and decisions led to a reliable reference. •

6. Prospects

During the project, the core developers proposed several follow-up interventions to the domain experts. We grouped these follow-ups into four areas: open-source readiness, enhancements to the machine-learning workflow, implementation of a logging library, and submission of the package to CRAN. Next, we briefly discuss these prospects and their potential benefits, roughly ordered by importance and ease of implementations.

Getting Open-Source Ready. To foster an open-source community, it is vital that a project is visible and easy to find. There are several ways to improve visibility. For instance, the project may be submitted to platforms that showcase different types of open-source projects. Additionally, Forest Foresight may gain exposure via blogs, news articles, and other media outlets. This paper also provides visibility for the project, and the domain experts are investigating other channels, too.

Besides increasing visibility, we stressed project maintenance and community building as key to retain newcomers to the project. As a means to facilitate onboarding, we have suggested to adapt labels or links for "good first issues," "help wanted," or "contributing" in the future. These labels are used by other projects and websites to provide starting points for newcomers and highlight different options to contribute to a project (Krüger et al., 2020; Tan et al., 2020).

Improving Machine-Learning Workflows. The core developers and domain experts agreed that the storing and management of machine-learning model versions with their features, parameters, metrics, and other artifacts could be improved. Ideally, different models could be exchanged to experiment, and rollbacks could be simplified. In addition, we identified collaborative model development with integrated data management as a possible direction. To address these needs, we proposed a transition to `MLflow`.[17] With our project, we laid a starting point for this endeavor and drafted possible strategies to move into this direction (cf. Table 1). Still, ultimately, this direction exceeded the scope of our project, requiring substantial training efforts to familiarize the WWF and other domain experts with `MLflow`.

Implementing Logging. During their work, the core developers noticed that it would be beneficial to introduce a logging library to obtain more informative bug reports. For them, the current lack of such a library hampered effective debugging. In addition to facilitating debugging, logging could also simplify adding reporting logs and customizing logging processes. We have recommended the R package `lgr`[18] as a feasible option. However, implementing logging within a project is an extensive task and the domain experts must first decide on their interests in this direction.

Introducing Forest Foresight to CRAN. Lastly, and most ambitious, we recommended submitting Forest Foresight to CRAN. CRAN is a popular package repository in the R community and being listed in it adds community visibility. Also, passing the CRAN review process represents a mark of credibility for end users and developers, potentially drawing in more contributors. In turn, the CRAN criteria are more ambitious than what we could implement within a single project, and than what the domain experts currently want to introduce. So, we suggested to keep an eye on CRAN and whether it becomes worth it to fulfill its criteria in the future.

At the end of our project, we reflected on the CRAN submission checklist[19] and the status of Forest Foresight. We found that we fulfilled many of the checklist's criteria, but some would require more engineering work and enough interest to maintain the status for future releases. In essence, Forest Foresight passed the `R CMD check -as-cran` continuous-integration pipeline without errors. However, we received a few warnings that would require further improvements of the package. Regarding the documentation, we added a `.Rd` file in the `man/` directory, a `README.md`- file, a `CONTRIBUTING.md` file, and documented exported functions using `roxygen2`[20] as required. Identically, we provide a correctly formatted `DESCRIPTION` file with title, description, and author fields filled in (but missing ORCID entries). Lastly, additional cross-platform testing would be required before submitting Forest Foresight to CRAN. Specifically, we tested Forest Foresight on Ubuntu and Windows, but not on macOS.

7. Threats to Validity

In this article, we reported the conduct, results, and experiences of a practical project, essentially representing an action-research-like experience report. Thus, our findings are inherently not generalizable to other software systems or developers. Instead, we focused on reporting our work and learnings in-depth for this specific case of Forest Foresight. While not generalizable, our work represents a substantial and unique case in which we introduced software engineering practices into an R package developed by domain experts. We hope that our experiences contribute supportive evidence to the broader body-of-knowledge and can help practitioners adopt software engineering practices.

Due to the lack of quantifiable metrics, we did not follow a full-fledged and systematic action-research methodology. For this reason, we cannot guarantee that the changes in the development of Forest Foresight can be attributed solely to our interventions. The intermixed introduction of our interventions is inherent to their nature and also more realistic, but challenges the internal validity regarding the impact of interventions. Still, we contribute experiences of a real-world case, and the feedback of the domain experts together with the clearly visible changes improves our confidence that our interventions had a positive impact.

8. Related Work

Researchers have extensively investigated open-source projects and communities. Throughout this article, we have referenced research related to our interventions, primarily on lowering barriers for contributors and newcomers (Fronchetti et al., 2023; Krüger et al., 2020; Steinmacher et al., 2019, 2015; Tan et al., 2020). More closely related, several researchers have investigated how to make open source projects more attractive and accessible to outside contributors. For instance, Kochhar et al. (2021) report the results of interviews on six Microsoft projects that transitioned from closed to open-source. Their findings highlight that even companies aim to build open-source

[17] https://mlflow.org/

[18] https://cran.r-project.org/package=lgr
DOI: 10.32614/CRAN.package.lgr

[19] https://cran.r-project.org/web/packages/submission_checklist.html

[20] https://cran.r-project.org/web/packages/roxygen2/index.html
DOI:10.32614/CRAN.package.roxygen2

communities and that the required changes for creating these communities impact development processes. In an older work, Kilamo et al. (2012) propose recommendations for releasing proprietary industrial software as open source and positioning it optimally for open-source contributions, which were used in four industrial case studies. Similarly, Sirkkala et al. (2009) have derived recommendations for the early steps of such transitions, which they evaluated through first insights into three case studies. Jansen et al. (2012) have developed a framework to judge how "open" a software-developing organization is. Lastly, Pinto et al. (2018) conducted a study on eight projects that moved from closed to open-source to investigate common beliefs. They found that it is challenging to attract and keep newcomers for such software projects.

Within the area of research software engineering (Lamprecht et al., 2022; Felderer et al., 2025; Cohen et al., 2021), several studies offer recommendations and rules on the sustainable development of open-source software by domain experts with varying experiences as software engineers. For instance, Ferenz et al. (2025) contribute recommendations for improving software developed by researchers without formal training in software engineering. With regards to fostering a healthy community, Prlić and Procter (2012) note that merely releasing the software under an open-source license is not enough if the goal is to encourage outside contributions. To mitigate this issue, the authors offer recommendations for developers who aim to encourage outside contributions to their scientific software.

Our work complements such previous research with insights into improving a different system, developed by a non-government organization using R for environmental protection. As such, we contribute a complementary case that is related to opening software for open-source communities and that can also inform research software engineering. Due to the different properties (e.g., industry versus non-government organization, domain of environmental protection), we provide novel insights as well as supportive evidence that interests and changes are similar across projects.

9. Conclusion

In this article, we shared our experiences of improving Forest Foresight's code quality, tooling, testing, processes, and documentation to prepare future open-source contributions. Besides insights into our changes, we reported particularly on teaching domain experts that do not have a software engineering background. Due to the unique nature of the project, we contribute an interesting case for how software engineering practices can improve a non-government organization's project. Still, this is only one step of a hopefully long-living software project that will continuously evolve and advance. So far, we focused on interventions that bring immediate value while being simple and demonstrable improvements. We plan to further improve Forest Foresight and its development in the future, aiming to establish a lively open-source community.

CRediT authorship contribution statement

Amin Bakhshi: Conceptualization, Methodology, Software, Validation, Investigation, Writing - Original Draft, Writing - Review & Editing, Project administration. **Hasrul Maruf:** Conceptualization, Methodology, Software, Validation, Investigation, Writing - Original Draft, Writing - Review & Editing, Project administration. **Maas van Apeldoorn:** Conceptualization, Methodology, Software, Validation, Investigation, Writing - Original Draft, Writing - Review & Editing, Project administration. **Zillah Calle:** Conceptualization, Software, Validation, Investigation, Resources, Writing - Review & Editing. **Jonas van Duijvenbode:** Conceptualization, Software, Validation, Investigation, Resources, Writing - Review & Editing, Project administration. **Ismay Wolff:** Writing - Original Draft, Writing - Review & Editing, Supervision. **Yanja Dajsuren:** Conceptualization, Writing - Review & Editing, Supervision, Project administration. **Jacob Krüger:** Conceptualization, Methodology, Writing - Original Draft, Writing - Review & Editing, Supervision, Project administration.

References

Beniamini, G., Gingichashvili, S., Orbach, A.K., Feitelson, D.G., 2017. Meaningful identifier names: The case of single-letter variables, in: International Conference on Program Comprehension (ICPC), IEEE. doi:10.1109/icpc.2017.18.

Binkley, D., Davis, M., Lawrie, D., Maletic, J.I., Morrell, C., Sharif, B., 2013. The impact of identifier style on effort and comprehension. Empirical Software Engineering 18. doi:10.1007/s10664-012-9201-4.

Bologna, M., Aquino, G., 2020. Deforestation and world population sustainability: A quantitative analysis. Scientific Reports 10. doi:10.1038/s41598-020-63657-6.

Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system, in: International Conference on Knowledge Discovery and Data Mining (KDD), ACM. doi:10.1145/2939672.2939785.

Cohen, J., Katz, D.S., Barker, M., Chue Hong, N., Haines, R., Jay, C., 2021. The four pillars of research software engineering. IEEE Software 38. doi:10.1109/MS.2020.2973362.

Daka, E., Fraser, G., 2014. A survey on unit testing practices and problems, in: International Symposium on Software Reliability Engineering (ISSRE), IEEE. doi:10.1109/ISSRE.2014.11.

Einhorn, C., Buckley, C., 2021. Global leaders pledge to end deforestation by 2030. The New York Times URL: https://www.nytimes.com/2021/11/02/climate/cop26-deforestation.html.

FAO, 2020. Global Forest Resources Assessment 2020: Main Report. Technical Report. Food and Agriculture Organization of the United Nations. doi:10.4060/ca9825en.

Felderer, M., Goedicke, M., Grunske, L., Hasselbring, W., Lamprecht, A.L., Rumpe, B., 2025. Investigating research software engineering: Toward RSE research. Communications of the ACM 68. doi:10.1145/3685265.

Ferenz, S., Frost, E., Schrage, R., Wolgast, T., Beyers, I., Karras, O., Werth, O., Nieße, A., 2025. Ten recommendations for engineering research software in energy research, in: International Conference on Future and Sustainable Energy Systems (e-Energy). doi:10.1145/3679240.3734606.

Fowler, M., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley.

Fronchetti, F., Shepherd, D.C., Wiese, I., Treude, C., Aurélio Gerosa, M., Steinmacher, I., 2023. Do CONTRIBUTING files provide information about OSS newcomers' onboarding barriers?, in: Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM. pp. 16–28. doi:10.1145/3611643.3616288.

Interpol, 2021. Forestry crime fact sheet 2021. https://www.interpol.int/en/content/download/17367/file/Forestry%20Crime%20Fact%20sheet%202021.pdf.

Jansen, S., Brinkkemper, S., Souer, J., Luinenburg, L., 2012. Shades of gray: Opening up a software producing organization with the open software enterprise model. Journal of Systems and Software 85. doi:10.1016/j.jss.2011.12.007.

Kilamo, T., Hammouda, I., Mikkonen, T., Aaltonen, T., 2012. From proprietary to open source—growing an open source ecosystem. Journal of Systems and Software 85. doi:10.1016/j.jss.2011.06.071.

Kochhar, P.S., Kalliamvakou, E., Nagappan, N., Zimmermann, T., Bird, C., 2021. Moving from closed to open source: Observations from six transitioned projects to github. IEEE Transactions on Software Engineering 47. doi:10.1109/TSE.2019.2937025.

Krüger, J., Hebig, R., 2023. To memorize or to document: A survey of developers' views on knowledge availability, in: International Conference on Product Focused Software Process Improvement (PROFES), Springer. pp. 39–56. doi:10.1007/978-3-031-49266-2_3.

Krüger, J., Mukelabai, M., Gu, W., Shen, H., Hebig, R., Berger, T., 2019. Where is my feature and what is it about? A case study on recovering feature facets. Journal of Systems and Software 152. doi:10.1016/j.jss.2019.01.057.

Krüger, J., Nielebock, S., Heumüller, R., 2020. How can i contribute? A qualitative analysis of community websites of 25 Unix-like distributions, in: International Conference on Evaluation and Assessment in Software Engineering (EASE), ACM. doi:10.1145/3383219.3383256.

Lamprecht, A.L., Martinez-Ortiz, C., Barker, M., Bartholomew, S.L., Barton, J., Hong, N.C., Cohen, J., Druskat, S., Forest, J., Grad, J.N., Katz, D.S., Richardson, R., Rosca, R., Schulte, D., Struck, A., Weinzierl, M., 2022. What do we (not) know about research software engineering? Journal of Open Research Software 10. doi:10.5334/jors.384.

Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. Journal of Systems and Software 101. doi:10.1016/j.jss.2014.12.027.

Mårtensson, T., Ståhl, D., Bosch, J., 2019. Test activities in the continuous integration and delivery pipeline. Journal of Software: Evolution and Process 31. doi:10.1002/smr.2153.

Martin, R.C., 2008. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

Martini, A., Besker, T., Bosch, J., 2020. Process debt: A first exploration, in: Asia-Pacific Software Engineering Conference (APSEC). doi:10.1109/APSEC51365.2020.00040.

Nielebock, S., Krolikowski, D., Krüger, J., Leich, T., Ortmeier, F., 2019. Commenting Source Code: Is It Worth It for Small Programming Tasks? Empirical Software Engineering 24. doi:10.1007/s10664-018-9664-z.

Pinto, G., Steinmacher, I., Dias, L.F., Gerosa, M., 2018. On the challenges of open-sourcing proprietary software projects. Empirical Software Engineering 23. doi:10.1007/s10664-018-9609-6.

Prlić, A., Procter, J.B., 2012. Ten simple rules for the open development of scientific software. PLoS Computational Biology 8. doi:10.1371/journal.pcbi.1002802.

Ripple, W.J., Wolf, C., van Vuuren, D.P., Gregg, J.W., Lenzen, M., 2024. An environmental and socially just climate mitigation pathway for a planet in peril. Environmental Research Letters 19. doi:10.1088/1748-9326/ad059e.

Robillard, M.P., 2019. Introduction to Software Design with Java. Springer. doi:10.1007/978-3-030-97899-0.

Runeson, P., 2006. A survey of unit testing practices. IEEE Software 23. doi:10.1109/MS.2006.91.

Schneider, D., Spurlock, S., Squire, M., 2016. Differentiating communication styles of leaders on the linux kernel mailing list, in: International Symposium on Open Collaboration (OpenSym), ACM. doi:10.1145/2957792.2957801.

Sirkkala, P., Aaltonen, T., Hammouda, I., 2009. Opening industrial software: Planting an onion, in: International Conference on Open Source Systems (OSS), Springer. doi:10.1007/978-3-642-02032-2_7.

Staron, M., 2020. Action Research in Software Engineering. Springer. doi:10.1007/978-3-030-32610-4.

Steinmacher, I., Gerosa, M., Conte, T.U., Redmiles, D.F., 2019. Overcoming social barriers when contributing to open source software projects. Computer Supported Cooperative Work 28. doi:10.1007/s10606-018-9335-z.

Steinmacher, I.F., Graciotto Silva, M.A., Gerosa, M.A., Redmiles, D.F., 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. Information and Software Technology 59. doi:10.1016/j.infsof.2014.11.001.

Tan, X., Zhou, M., Sun, Z., 2020. A first look at good first issues on GitHub, in: Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM. doi:10.1145/3368089.3409746.

Wickham, H., 2011. testthat: Get started with testing. The R Jounral 3. doi:10.32614/RJ-2011-002.

Wickham, H., Bryan, J., 2023. R Packages: Organize, Test, Document, and Share Your Code. O'Reilly.

Wolff, N.H., Vargas Zeppetello, L.R., Parsons, L.A., Aggraeni, I., Battisti, D.S., Ebi, K.L., Game, E.T., Kroeger, T., Masuda, Y.J., Spector, J.T., 2021. The effect of deforestation and climate change on all-cause mortality and unsafe work conditions due to heat exposure in berau, indonesia: A modelling study. The Lancet Planetary Health 5. doi:10.1016/S2542-5196(21)00279-5.

WWF, 2022. Forest foresight prospectus. https://www.wwf.nl/globalassets/pdf/forest-foresight/wwf-forest-foresight-prospectus.pdf.

Zeller, A., 2009. Why Programs Fail - A Guide to Systematic Debugging. Academic Press.