FISHER: A Sustainable Software Procurement Framework

Valentijn van de Beek, Merlijn Mac Gillavry, Leon de Klerk, Joey de Water

Delft University of Technology

March 2023

Abstract

Green IT has become an increasingly relevant and impactful topic in the domain of Software Engineering. Nowadays, software not only needs to be efficient, but also sustainable. In a world where energy and resources used in computers are becoming more scarce and where climate change is one of the most pressing issues facing our species, research needs to be done on how to make IT more sustainable. An important component of making sustainable software more adopted by governments and corporations is the procurement of green software. Therefore, in this paper, we present FISHER, an experimental framework based on earlier research in sustainable software that helps people and corporations with sustainable software procurement. Our framework is based on three categories, with criteria that extend beyond the basics of the sustainability of the software itself and also focus on the social sustainability of the company itself. Additionally, we provide a basic implementation of our framework as a Proof of Concept (POC), that can already be used for sustainable software procurement.

1 Introduction

Green IT has become a hot topic in software engineering research over the past decade as sustainability has entered the zeitgeist. Often, research has focused on the impact that a particular piece of software has on the environment, or, based on various metrics and guidelines, on how to reduce the energy usage of software. However, all of this research has been done from the perspective of the software developer and, therefore, assumes a level of access that is not available to the buyer of software. Parallel to this development is the rise of strategies for general sustainable procurement, where companies, governments, and NGOs take sustainability into account while making purchasing decisions. For example, by measuring the social, environmental, and economic impact of planting a cacao farm in a more prosperous country rather than one with lower wages.

Though there has been a rise in sustainable procurement for other domains, in terms of sustainable software procurement little work has been done. Therefore, we ask the following research question: "What are the methods and criteria that can be used to evaluate the sustainability of a software solution, with the goal of facilitating sustainable software procurement practices?". In this paper, we focus on defining an initial framework for green software procurement. As a result, we introduce FISHER, *FoundatIon Software Health mEtRics*, a new framework where the three aspects of sustainable measurement are brought together. With this, we aim to provide a basis for buyers to make sustainable sound decisions when buying software, and provide key points for software suppliers to improve the sustainability of their business and products.

The contributions of this paper are as follows:

- Combining software health suggestions with concrete measurements.
- Adding foundational business constraints into procurement.

- Formalizing the 3Cs of questions: concrete, conditional and consideration.
- An example implementation of the framework using web-based technologies.

2 Related Work

Surprisingly, the subject of sustainable procurement in software engineering has received little to no attention in literature. Sustainable procurement is typically used in relation to physical projects and, to a limited extent, to the purchasing of sustainable hardware. Green software is typically discussed from a supply-side perspective rather than the demand-side. This means that prior research typically assumed more control over the internals of the software than would be possible during a procurement procedure.

2.1 Green labelling

An interesting field of research is the consumer-oriented green labelling of sustainable software products. In [Kern et al., 2015], criteria for sustainable software products, label representation, target groups and stakeholders are identified. They propose criteria and suitable representations for a sustainability label using these aspects. The main takeaways are the resource-oriented and well-being-oriented feasibility, with criteria like energy consumption and accessibility. [Naumann et al., 2021] is an implementation of the aforementioned criteria into a sustainability label developed by the German government.

Though these strategies are related to the sustainable procurement process, they cannot be applied directly, as they are targeted at finished products and have a focus on the end-user instead of a buyer. Blue Angel is used to label existing software, while FISHER can be used for in-process or planned software. It is also an iterative approach with intermediate feedback, while Blue Angel is applied once. The main relevant takeaways from these works are the criteria for resource efficiency, providing insight in the different categories of software sustainability.

2.2 Energy consumption

One factor that can be overlooked is the influences on energy consumption programming language that is used. In [Pereira et al., 2017], different software languages are analyzed. They look at runtime, memory usage, and energy consumption of twenty-seven well-known languages, most notable are Python, C, and Java. The differences between languages have measurable impact on the final product, where the same functionality can result in vastly different energy and memory usage depending on the language used. In extreme circumstances, this can be a difference of 760%.

2.3 Software and organization sustainability

In [Alsayed and Deneckère, 2022], a sustainability matrix for smartphone applications is proposed. This sustainability matrix is used to identify the degree of sustainability and to give project managers a way to test the sustainability of their smartphone applications. The matrix is based on interviews with experts, a questionnaire and a literature study. It consists of two categories, namely conception criteria and development criteria, which have twenty criteria in total.

Instead of focusing on concrete health metrics, [Lami and Buglione, 2012] focused on iterative improvements of certain aspects of the software. This already marks a shift from just software requirements, to sustainability integrated into the software development workflow. [Gallagher, 2016] takes this one step further and looks at the principles of sustainability on an organizational level, instead of a software level. Integration of sustainability principles at the core of the company itself is key to the longevity of sustainable principles within the organization and its products.

FISHER integrates these into a singular framework but changes the perspective. While the previous research focused on the company implementing the software, our framework instead is focused on the

purchaser. This means direct measurements are less feasible and that the only control over the outcome is based on the set requirements.

2.4 Software Procurement

In [Zhou, 2020], the author identifies three common mistakes made in software:

- Using technical measurements while the software does not yet exist.
- Upgrades to existing software are hard to cast into public offers due to differences in technical capacities by suppliers.
- Software often has integrated dependencies which need to match, e.g. which operating system is used

The author then argues for making the functional requirements the cornerstone of software procurement and that new software must be procured using a public offering. On the contrary, when upgrading software, a preference must be given to the original developers.

[Sieverding, 2008] argues for a neutral IT procurement strategy which focuses on objectively measurable criteria such as interoperability, cost and quality of life considerations. Favoring a particular social aspect like Free/Libre software is illegal in some jurisdictions and could lead to an overall worse or more expensive product. This differs significantly from FISHER, which aims to support social aspects by finding objective measurements based on industry best practices. In the view of the authors, the legal concerns with preferential treatment to these concerns are counterbalanced with new European directives with regard to reducing energy consumption [Commission, 2018], however, the authors acknowledge that additional research into the legal aspects of such a strategy within the European context remains an open question.

As argued in [Hochstetter et al., 2022], the transparency of procurement is vital to combating corruption in government. Here they develop a maturity model where they list five stages of procurement from the least transparent (1) to the most transparent (5). The main differences are taking an iterative approach in procurement and making use of open criteria.

3 Theoretical Sustainability Framework for Sustainable Software Procurement

We propose the FISHER framework as a guideline for sustainable software procurement. The framework consists out of the three categories summarized in Table 1, namely *Social Sustainability*, *Software Sustainability* and *Measurable Metrics*, which contain evaluation criteria that can be used to identify the degree of sustainability of a supplier and their software. *Social Sustainability* is a category unique to FISHER and reflects long-term business characteristics that are typically not considered in other frameworks. The latter two are derived from the prior work done in the fields of green software labelling and Green IT development. The specific criteria of each category are discussed in section 4.

Feature	Description
Foundational	Long-term characteristics of the organization creating software
Software Health	Suggestions that can improve the sustainability of a software package
Metrics	Measurable aspects of program execution of a critical use-case

Table 1: Overview of each category: social, software, and measurable metrics

3.1 Social Sustainability

Other frameworks do not take the social aspect into account and the focus of prior literature is on evaluation of the sustainability of a particular package [Mehra et al., 2022, Debbarma and Chandrasekaran, 2016] rather than comparing between a set of packages. For a package, the software and development strategies

are relevant factors, but properties of the company or community building the software is treated as a given. However, this aspect of procurement is not something that should be easily glossed over. Institutionalizing sustainable thinking in an organization is a crucial step in ensuring that any product created by that organization will be, and remain, sustainable over the lifespan of the product [Gallagher, 2016]. The creation of the software is typically a small period of the total lifespan, which may, in some circumstances, last multiple decades after initial creation. Examples of this are the Linux Kernel, airplane ticketing systems, or bank systems. 50% of a programmer's time is spent on debugging the program. [Britton et al., 2013]. Therefore, it is important any organization is not sustainable just for the duration of the procurement process, but also beyond that. Otherwise, you risk a backslide where the product becomes less sustainable due to inefficient, slow or badly executed maintenance.

3.2 Software Metrics & Improvement

Simply looking at the social aspects of an organization is not necessarily enough to determine whether a product is sustainable or not. It might be that the product under-performs, is made using wrong fundamentals, or that there simply is a better product available. Important here is to not only look at *how* a package performs, but also *why* it does. Looking only at the former might lead you to choose a suboptimal product, give incoherent feedback, or create impossibly complex guidelines. Criteria about which data structure, algorithm, or caching strategy are applied to do not naturally lend themselves to being reduced to single optimal short-form questions, but are still critical to the performance of the product [Naumann et al., 2015]. For example, an inventory management application that sorts items using Quicksort will vastly outperform one using Bubblesort, even if the latter has a better score in terms of concrete actions.

It is important to take a look at how metrics could be measured for a software package. One program may include many features or execution paths which may be used infrequently, often referred to as the 90-10% rule. This refers to the rule of thumb that 10% of the instructions are executed 90% of the time, while the rest are only rarely used [Smith, 1986]. An example of this are browsers, although browsing the web is the main functionality, it contains many more features such as theming, extensions, keeping track of videos being played, history, bookmarks, and more. These features are only used occasionally, but do not constitute the typical usage of the browser. Therefore, these are less important when attempting to measure a representative use case of the product.

3.3 Energy comparison using critical use-cases

Given that the framework targets procurement procedures, a set of requirements for the product has already been created that is given to suppliers for their procurement bid. Out of these requirements, it is possible to find a *critical use-case,* which is a set of requirements that encompass the expected most common use-case of a product. For example, for a mail client, these could be the requirements: reading an email, sending an email, and synchronizing with the mail server. This *critical use-case* is a common set of features that should be available across all suppliers, allowing for a procurement bid Proof of Concept implementation. This POC implementation can be used to compare the energy usage between the different suppliers and is indicative of the energy usage of the final software.

Another option to determine energy usage, would be to compare all code paths. Though, this is suboptimal since it would penalize implementations that offer more features and favor those that aim to minimize the average energy usage. It is also possible to compare a package to a large database of previously measured software of a similar kind, however, in procurement the aim is to find the best of the available options. It is therefore not necessary useful to compare to a global best, since the procurement still needs to be full-filled.

3.4 Positive feedback loop

The framework categories are not independent of each other, but rather are contained in a virtuous cycle (see Figure 3.4) where each is necessary but not sufficient. Ignoring one of these aspects can cause a problem which might negate the benefits of the others. By not including the social aspects, a company might be tempted to revert to older non-sustainable practices due to a lack of top-level oversight. If, on the other hand, software criteria are ignored, the system becomes a black box, which makes it more difficult to understand the resulting metrics, and make it more difficult to make improvements. Finally, the metrics allow for concrete comparison between the software packages and measure elements of the system, which can uncover problems with the software criteria and help improve them. The strengths and drawbacks of each category are given in Table 2.

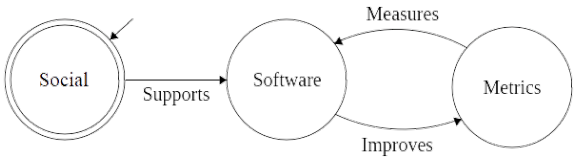

Figure 1: Diagram of FISHER stages

The categories are different stages that link into one another. Social criteria relate to whom you are doing business with and define the long-term priorities of the organization. Software health are concrete elements of the system and help to improve metrics, which can be positively extended and iterated on over time. This provides an incentive structure that not only rewards the best product, but also encourages a movement of systemic change.

Combination	Strength	Drawback
Social + Software	Best practices from trusted partner	Suboptimal choice due to hidden variables
Social + Metrics	Best product from trusted partner	No improvement suggestions
Software + Metrics	Best product following best practices	Sustainability may decrease over time

Table 2: Drawbacks of each category combination

3.5 Cross-cutting question variants

Out of prior work, specifically [Alsayed and Deneckère, 2022, Kern et al., 2013, Lago et al., 2014] three general questions variants were distilled: **concrete**, **conditional** and **consideration**. We refer to these three question types as the 3Cs. The category criteria are associated with the question types, and each question type can occur in any category. For example, criteria from the metrics category are consideration questions, as they do not have a clear good or bad result. NP-Complete problems, for example, simply will take a lot of resources and having a package that claims to solve such a problem at a very high efficiency even might be an indication of a fault.

Concrete questions are the primary type and refer to statements whereby there is a clear good or bad answer. One example is the criterion "Style guide". Since it is strictly better to use a style guide compared to no style guide, the system can make a concrete recommendation to the user. This recommendation can be augmented by additional explanation which explains why this question is relevant or when it might be better to ignore the suggestion. For this example, it could be: *A style guide is a sign of better code health.*

Conditional questions are used when a criterion is only applicable in a specific scenario. For example: *Light/dark mode.* For programs without a UI, this question would not be relevant and therefore is seen as not applicable. But for a project with UI requirements, it could be a useful metric.

Consideration questions do not necessarily have a good or bad answer, but should be considered when evaluating products. They can either be concrete considerations between a set of options or open questions. An example of the former would be based on the criterion *Private, public or mixed cloud* and for the latter *Number of bytes*. Neither of these questions have a distinct correct answer, as sometimes the privacy of a private cloud is preferable to the efficiency of a public one, but are options that should be considered. Here, the key is finding general questions which might be overlooked otherwise, which can make a difference in the decision-making process.

4 Criteria of the FISHER framework

This section lists all the current framework criteria, per category. These criteria form the basis of each category and are the actual aspects that can be used for the procurement process. Additionally, all criteria are formatted according to one of the framework question types. The framework currently treats all categories and criteria as equally important and does not provide a numerical value to answers.

4.1 Social Sustainability

This category consists of eight criteria relating to the business practices of the supplier. This includes business identity, long-term improvements and other typical procurement criteria. For all criteria, there is a reason for the importance and meaning of the criteria.

- **Accessibility features**. Accessibility is a large pillar of inclusiveness. For social sustainability, it is essential to make software accessible to as many users as possible.
- **Sustainability guidelines**. Having a sustainability policy helps employees deal with sustainability. As a company, it demonstrates its commitment to environmentally positive, social and ethical practices.
- **Sustainable alliance or group**. Being part of a sustainable alliance or group shows the company is already taking steps to increase its sustainability. This is another demonstration of commitment to environmentally positive practices.
- **Privacy, diversity and sustainability officer in C-Suite** Embedding the principles of privacy, diversity and sustainability at a high level at the company is a good way to ensure that these are actually upheld. This metric is not relevant for startups or small companies.
- **GDPR compliance.** GDPR compliance ensures user privacy, which is another pillar of social sustainability.
- **B-corp and other certifications.** B-Corps is a certification given to companies that uphold the principles of social and ecological sustainability.
- **Grid load and energy mix.** As a company, taking the grid load and energy mix into account can help reduce the carbon intensity of the power grid. It also shows that a company is aware of the impact.
- **Green strategy.** A green strategy allows a company to help make decisions that have a positive impact on the environment. As with sustainability guidelines, a green strategy also demonstrates the company's commitment to sustainability.

4.2 Software Sustainability

This category consists of fourteen criteria related to the best practices in software development that can lead to lower energy consumption. These metrics allow projects to improve their products, as these points can be acted upon. They can also be seen as guidelines for non-experts. In this category, the following criteria can be found:

- **Programming language.** Some programming languages take longer, more memory, and therefore more energy to solve specific problems than others. This criterion is assessed according to the energy consumption of each language in [Pereira et al., 2017].
- **Compiled or interpreted.** Interpreted programming languages are easier to write, but require more energy to run than using compiled programming languages.
- **Native or web-based.** This criterion applies to both desktop and mobile applications. Web-based applications are cross-platform and allow for easy deployment across multiple operating systems, while native applications are typically highly dependent on OS. However, a native application requires less energy since it does not depend on a personal browser instance to run. Choosing a web-based framework could make sense for a private, low-user application which is used by a heterogeneous group of users.
- **Cached requests receiver.** This criterion applies to networked applications. Sending redundant information over the web requires energy, so if it is possible, non-single-use data should be cached on disk.
- **Private, public or mixed cloud.** This criterion applies to networked applications. A private cloud is more secure and private, but is less sustainable than a public cloud. This choice is highly dependent on the organizational requirements.
- **Open source licensing.** Free software allows for modification of the source code, use by the broader community, and can serve as a hedge for the company going bankrupt. Copyleft requires changes to be shared under the same license, while permissive does not. Typically, having an open source license is a net positive, but the licensing choice may depend on the project.
- **Lazy loading.** This criterion applies to web-based applications. Lazy loading allows a web-based page to render, request, and push content at a later time. It contributes to the efficiency of operations and decreases energy consumption at runtime.
- **Background work.** Performing non-essential tasks in the background allows for better utilization of the computer resources since it makes use of all cores, and it also gives a better user experience.
- **Virtualization.** This criterion applies to non-mobile applications. Virtualization for better isolation between processes increases security. It also allows for tighter management of the resources used by the application, increased isolation, and more efficient utilization of resources.
- **Light/dark mode.** This criterion applies to UI-based applications. Dark mode consumes less energy than light mode, especially on OLED devices where the pixels are actually physically turned off.
- **Style guide.** By following a style guide, a project has a consistent coding style throughout the project, which is a sign of better code health.
- **Static analysis.** Static analysis tools scan the project for any faults, security issues or style issues. This makes sure that code health is maintained, and fewer bugs appear in the final product.
- **Project certifications.** Some companies and alliances provide certifications that indicate that the project has a certain quality standard. A classical example of this is MISRA for embedded devices used in automotive, aerospace and critical infrastructure.
- **Security audits.** All software has bugs, by frequently auditing the software for vulnerabilities problems may be found much earlier and catastrophic failure can be avoided. This is relevant for large or mission-critical applications.

4.3 Measurable Metrics

This category contains criteria specifically related to the function of the software. It gives an opportunity to gain concrete measurements on applications. The criteria allow for an apples-to-apples comparison, specifically measuring a critical-path use case. An extra advantage is that these criteria detect hidden variables, e.g. chosen algorithm, hardware attributes and data storage.

- **Number of bytes.** This criterion measures the number of bytes sent over the network.
- **Network requests.** This criterion measures the number of network requests made.

- **CPU usage.** This criterion measures how many CPU cycles are being used by the project.
- **Hard drive accesses.** This criterion measures how many times the hard drive is accessed.
- **Energy consumption.** This criterion measures how much energy is used by the application.

5 Experimental Design

In line with the framework proposed, we created a proof of concept tool based on the framework. This was done to provide an example of how the framework would be applied in a real-world setting. Initially, we envisioned a tool where a client could set up projects and add different companies to this project. The framework could then be used to gather information about the companies in regard to the project and allow for a comparison between them. Due to time constraints, we reduced our tool to a standalone form, where a client can fill out the questionnaire and show results. The questions, categories and results are all based directly on the framework, although some were altered to better fit the format of the tool. The questionnaire can be found in appendix A. The rest of this section will discuss the design and choices made for the general framework implementation and both types of tooling.

5.1 Framework implementation

For the proof of concept, we created a web-based tool that implements the categories and criteria defined in the framework. In this section, we shortly discuss the technology used and how the framework was translated into a tool.

5.1.1 Tech stack

The web-based tool is built as a single-page application written in TypeScript and Vue.js, for styling the Bulma CSS framework was used. The use of a component-based framework allowed for re-use of the same elements on multiple pages. These components are all custom, as it allowed for a consistently styled set of components that is tailored to the requirements needed to implement the framework.

The backend of the experimental implementation is based on the article written by *Emen* on *Codevoweb.com* [1]. It's a multi-container docker system using a TypeScript based Node.js application using the Express API framework, a MongoDB database for storage, and a Redis cache for session storage.

5.1.2 Implementation

For the tool, all the categories from the framework were used and, in addition, the criteria of the framework were converted to questions that are representable on a computerized form. This allows for a direct mapping of the criteria into the tool, except for criteria in the Measurable Metrics category, which differ in the standalone system from the framework criteria as is explained in section 5.2. The difficulty of representing the framework was in how to define the answers, how to score the answers given, and how to display the results. For each question four possible answer types were defined in the implementation:

- Input, a free input field where every answer can be filled in
- Boolean, a selector that provides only yes and no as an option
- Selector, a selector where the options are defined per question
- Dropdown, a dropdown which is similar to the selector but used for questions with more than four options.

The mappings between the types directly relate to the 3C question types of the framework. All concrete questions are defined as a boolean type, while conditional questions use the selector type. Consideration questions use the boolean type for the more concrete questions, the selector type for questions with multiple

[1] https://codevoweb.com/node-typescript-mongodb-jwt-authentication

options, or the input type for open questions such as the metrics. For the conditional questions and some consideration questions, we define the *Not Applicable* option, which eliminates the question from the questionnaire results if selected. This is done as not every question is relevant to the requirements of the project, and therefore allows for a level of customizability. For questions with more than four options, a dropdown is used instead of a selector, such as for the programming languages.

In the current proof of concept, each question and category has an equal weight. The answers considered more sustainable by the framework criteria are given a score of 1, while the less sustainable answer is given a score of 0. This discrete mapping is applied to almost all questions, with two exceptions. The question about compiled or interpreted code contains a middle-ground option *balanced*, which has a score of 0.5. Balanced in this context refers to technologies that fall between compiled and interpreted, such as JIT or mixed technologies such as CPython. Additionally, the question about programming languages associates a fractional score between 0 and 1 for each language based on their position in table 4 of [Pereira et al., 2017].

The results themselves are grouped per category, where the question, answer, and score are displayed. Together with the results, if available, each question also states the clarification. The clarification gives extra information about the question and can help to interpret the answers for both buyers and suppliers. The scores are shown per category as a percentage, which allows for an easier comparison. The maximum score, 100%, of each category is determined as the total number of questions that were not answered with *Not Applicable*. The aggregated results are extended with a clarification of the meaning, this includes a general explanation of how scores should be interpreted, and extra details on what a high or low score means for each category. This extra information is based on the meaning of the different categories and is fully based on the framework itself.

5.2 Standalone system

The working proof of concept is a standalone system. This means that it provides a page containing the questionnaire and a page displaying the results. In the standalone version, there are no other entries to compare to, and therefore it is impossible to score the Measurable Metrics answers. For this reason, all criteria from the Measurable Metrics category have been changed from an open consideration question to a concrete consideration question. Instead of asking for the measured values of the system, the question asks if such metrics are measured at all. In this way, the criteria can still be used, although differently compared to the framework. The main use for this system is that a buyer can quickly get an indication of the sustainability of a company, or a company can get an indication of their own performance.

5.3 Integrated system

Although the integrated system does not have a completed proof of concept, it is worth mentioning the differences compared to the standalone system. The integrated system requires a client to set up a project and add different companies to it. For each company, the questionnaire can be filled out either directly by the client or by a representative of the company. In the current proof of concept, there is support for user authentication, project creation, and company creation. Though it lacks persistent data, an option to answer questions, and a way to see the results.

6 Future Work

In this section, we discuss the current limitations of the framework and the POC. We propose ideas on how to improve, extend, and validate our current work.

6.1 Framework

For the framework, there are three areas that could be extended. The first point is project-specific criteria. Each application type, e.g. desktop or mobile, could have its own criteria. This will result in more accurate degrees of sustainability for each project type and corresponding company type, making it useful for all types of sustainable software procurement.

The second point is related to extending the Measurable Metrics category. Energy consumption data of different projects and critical-path use cases can be gathered and stored in a database. This data can then be used to compare projects with other roughly equivalent projects instead of just comparisons within a project, giving more insights into how a company performs and what it can improve upon.

The final point is physical software product criteria. Some software products are distributed using physical products, e.g. CDs, or have associated physical elements. These physical aspects do not directly impact the sustainability of the software, but are a part of the project in general and should therefore be taken into account during the procurement process.

Additionally, the framework currently lacks real-world validation. Though it is shown that the framework can be implemented into a working application, we recommend future work to focus on validating the questions and categories with the different stakeholders.

6.2 Implementation

The implementation should be extended on two main points. First, the general system could be enhanced by introducing more refined scores and category weights. This allows for a more fine-grained evaluation of different companies based on the requirements of a specific project or context.

The second point is to complete the integrated system. This means that all data is persistently stored in the database. This would make it possible to add projects, fill out the details for each company, and most importantly show the results. Building on the first point, the integrated system could expand the results and should allow for comparisons between the different results within a project. This would provide an example that is better applicable in a real client setup, where they can compare the different companies and their bids. In addition, there are some other aspects that could be interesting to implement for future works:

- Configurable forms, where the client can specifically define the questions for a specific project based on the available questions of the framework and the context of the project.
- Form sharing, instead of having to gather details about a company, part of the form could be sent to the company. It would then need to be filled out as part of the procurement bidding process.
- Answer verification, we envision a process where an external third party can see and verify the results. This would increase the trustworthiness of the results and allow for fairer and more correct comparisons.

To ensure that future work can build upon the basic implementation we present in our paper, we have made the repository public[2] and provided a *README* file that explains how to set up and run the project.

7 Conclusion

At the time of writing, there has been little research done in the field of sustainable software procurement. In this paper, we introduce a new framework, FISHER, which serves as a guideline for sustainable software procurement. It builds on existing work done in the sustainable software domain to provide three categories that impact sustainability. The categories each define a set of criteria that are related to the sustainability domain of the category. These criteria are used to validate a company and its software against the defined sustainability metrics. This allows a client to score and compare different options in a procurement process

[2]https://github.com/leondeklerk/SSE/tree/master/project_2

on short and long-term sustainability aspects. Based on our theoretical framework, we developed a web-based implementation that shows that the framework can be applied to a practical product. Though in its current state, the framework is still experimental and has not been externally validated with end-users or in a real-world scenario, it could be used as a stepping stone for deriving accepted standards in the software industry. With this framework, we aim to assist with sustainable software procurement and reduce the environmental impact of the software used in our society.

References

[Alsayed and Deneckère, 2022] Alsayed, A. and Deneckère, R. (2022). A sustainability matrix for smartphone application. In Horkoff, J., Serral, E., and Zdravkovic, J., editors, *Advanced Information Systems Engineering Workshops*, pages 73–85, Cham. Springer International Publishing.

[Britton et al., 2013] Britton, T., Jeng, L., Carver, G., Cheak, P., and Katzenellenbogen, T. (2013). Reversible debugging software. cambridge judge business school.

[Commission, 2018] Commission, E. (2018).

[Debbarma and Chandrasekaran, 2016] Debbarma, T. and Chandrasekaran, K. (2016). Green measurement metrics towards a sustainable software: A systematic literature review. In *2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, pages 1–7.

[Gallagher, 2016] Gallagher, D. R. (2016). Climate change leadership as sustainability leadership: From the c-suite to the conference of the parties. *Journal of Leadership Studies*, 9(4):60–64.

[Hochstetter et al., 2022] Hochstetter, J., Díaz, J., Diéguez, M., Espinosa, R., Arango-López, J., and Cares, C. (2022). Assessing transparency in egovernment electronic processes. *IEEE Access*, 10:3074–3087.

[Kern et al., 2015] Kern, E., Dick, M., Naumann, S., and Filler, A. (2015). Labelling sustainable software products and websites: Ideas, approaches, and challenges.

[Kern et al., 2013] Kern, E., Dick, M., Naumann, S., Guldner, A., and Johann, T. (2013). Green software and green software engineering - definitions, measurements, and quality aspects.

[Lago et al., 2014] Lago, P., Gu, Q., and Bozzelli, P. (2014). *A systematic literature review of green software metrics*. VU Technical Report.

[Lami and Buglione, 2012] Lami, G. and Buglione, L. (2012). Measuring software sustainability from a process-centric perspective. In *2012 Joint Conference of the 22nd International Workshop on Software Measurement and the 2012 Seventh International Conference on Software Process and Product Measurement*, pages 53–59.

[Mehra et al., 2022] Mehra, R., Sharma, V. S., Kaulgud, V., Podder, S., and Burden, A. P. (2022). Towards a green quotient for software projects. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 295–296.

[Naumann et al., 2021] Naumann, S., Guldner, A., and Kern, E. (2021). The Eco-label Blue Angel for Software Development and Components. In Kamilaris, A., Wohlgemuth, V., Karatzas, K., and Athanasiadis, I. N., editors, *Advances and New Trends in Environmental Informatics*, Progress in IS, pages 79–89. Springer.

[Naumann et al., 2015] Naumann, S., Kern, E., Dick, M., and Johann, T. (2015). Sustainable software engineering: Process and quality models, life cycle, and social aspects. In *ICT Innovations for Sustainability*, pages 191–205. Springer.

[Pereira et al., 2017] Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J., and Saraiva, J. (2017). Energy efficiency across programming languages: how do energy, time, and memory relate? pages 256–267.

[Sieverding, 2008] Sieverding, M. (2008). Choice in government software procurement: A winning strategy. *Journal of Public Procurement*, 8(1):70–97.

[Smith, 1986] Smith, C. U. (1986). Independent general principles for constructing responsive software systems. *ACM Trans. Comput. Syst.*, 4(1):1–31.

[Zhou, 2020] Zhou, T. (2020). The difference between software and hardware and the determination of procurement methods. In *3rd International Conference on Advances in Management Science and Engineering (IC-AMSE 2020)*, pages 47–51. Atlantis Press.

A Sustainability Questionnaire

A.1 Social Sustainability

- *Does the company implement accessibility features?*
- *Does the company have sustainability guidelines?*
- *Is the company part of any sustainable alliance or group?*
- *Does the company have a privacy, diversity and sustainability officer in the C-Suite?*
- *Does the company ensure GDPR compliance?*
- *Is the company a B-Corp? Does it have any other certifications (ex. ISO)*
- *Does the company take grid load and energy mix into account?*
- *Does the company have a green strategy?*

A.2 Software Sustainability

- *Which programming languages are used?*
- *Is the program compiled or interpreted?*
- *If a desktop or mobile app, is it native or web-based?*
- *If a networked app, are requests cached on the receiver?*
- *If a networked app, does it use a private, public or mixed cloud?*
- *Is it proprietary, copyleft or permissive?*
- *If a web-based app, does it lazy-load assets?*
- *Are non-essential tasks done in the background?*
- *If not a mobile app, is it virtualized?*
- *If UI-based, is there a dark-mode?*
- *Does the company have a style guide?*
- *Is static analysis applied to the code?*
- *Which, if any, certifications (ex. MISRA) does the project have?*
- *Is the software subject to regular security audits?*

A.3 Measurable Metrics

- *What are the number of bytes sent?*
- *What is the number of network requests made?*
- *How many CPU cycles are being used by the project?*
- *How many times is the hard drive being accessed?*
- *How much is the energy usage of the software?*