Sustainable Software Engineering Paper

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

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 in terms of used resources and time, but it also needs to be more efficient in terms of energy consumption. 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 green. 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 find a green solution for their IT problems. Additionally, we provide a basic implementation of our framework as a Proof of Concept (POC), that can already be used for green software procurement.

1 Introduction

Green IT has become a hot topic of 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 based on various metrics and guidelines on how to reduce that energy usage. 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 are strategies around green procurement whereby companies, governments and NGOs take sustainability into account while making purchasing decisions. For example, by measuring the social, environmental and economic impact of deciding to plant a cacao farm in a more prosperous country rather than one with lower wages.

In the literature various strands of research can be discovered. You can view the field of sustainable software from the lens of either concrete health metrics (as done in [Alsayed and Deneckère, 2022]) or a set of metrics that be improved on by iterative improvement of certain aspects of a program (as done in [Lami and Buglione, 2012]). One aspect of sustainability is often overlooked, namely whether the principles of sustainability are integrated into the organisation developing the software. Support from the top is the key to the longevity of sustainable principles within the organisation [Gallagher, 2016].

This paper introduces FISHER, *FoundatIon Software Health mEtRics*, a new framework where the three aspects of sustainable measurement are brought together. Key for the framework

is that aspect each builds upon another aspect of software development and combines in such a way as to create a virtuous cycle that promotes the longevity of sustainable principles.

Contributions of the paper are as follows:

- Combining software health suggestions with concrete measurements.
- Adding foundational business constraints into procurement.
- Formalising the 3Cs of questions: concrete, conditional and consideration.
- An implementation of the framework on web technologies.

2 Related Work

2.1 Green procurement in software engineering

Surprisingly, the subject of green procurement in software engineering has received little to no attention in literature. Green procurement is typically used in relation to physical projects and, to a limited extent, to the purchasing of sustainable hardware. Software is only discussed from a supply-side perspective rather than that of a demand-side, meaning that prior research typically assumed greater control over the internals than would be possible from a procurement procedure. One field of research that is very close to that of this paper is the consumer-oriented green labels for software which aim to communicate the sustainability of a piece of software by having it checked by an external governing agency. This differs from the problem in this paper since this can only be done after the project is finished, assumes little control over the requirements, and is focused on the end-user. Procurement is applied at the start of the project and focuses on a larger organization with control over the requirements.

2.2 Green software 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. Finally, [Naumann et al., 2021] presents another sustainability label. They introduce software products as a new product group for the "Blue Angel" eco-label, a German product-related environmental label that is awarded to environmentally friendly products and services, and propose criteria for the label. The main takeaways are the criteria for resource efficiency. Both these sustainability labels provide us with insights into the different categories which can be taken into account for our framework.

2.3 Software energy consumption

A source that might be overlooked that influences energy consumption is the programming language that is used. In [Pereira et al., 2017], different software languages are analysed. They look at runtime, memory usage and energy consumption of twenty-seven well-known languages, most notable

are Python, C and Java. We use the comparisons of energy consumption for each programming language to give each language a score in our proposed framework.

2.4 Sustainability matrix

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. The criteria we extracted for our framework are *compiled vs interpreted*, *lazy loading*, *background work*, *night/day mode*, *optimize the use of the CPU* and *Percentage open source use*.

3 Sustainability Framework for Sustainable Software Procurement

We propose the FISHER framework for sustainable software procurement. The framework consists of three categories, namely *Social Sustainability*, *Software Sustainability* and *Measurable Metrics*. The categories contain criteria, which can be used to identify the degree of sustainability of a supplier. The latter two correspond to prior work done in the field of, respectively, software green labelling and software Green IT. This is contrasted with that of the *Social Sustainability* which is often not considered during considerations of a software product.

Feature	Description
Foundational	Long-term characteristics of the organisation 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

3.1 Social Sustainability

Not taking the social aspect into account is likely due to the fact that most literature focuses on finding the sustainability of a particular package rather than that of a set. Hereby are the software and development strategies relevant factors, but is that of the company or community building software treated as a given. However, this aspect of procurement is not something that should be glossed over or forgotten. Institutionalising sustainable thinking in an organisation is a crucial step in ensuring that any product created by that organisation will be and, critically remain, sustainable over the lifespan of the product. The creation of software is typically a rather small period of the total lifespan which may, in some circumstances, last multiple decades after initial creation. Common wisdom within the field is thus that a typical developer will spend a fraction of the time writing code that they will spend on reading code. Therefore, it is important any organisation is not sustainable just for the duration of the procurement, but also beyond that. Lest you risk a slow backslide of the initial goals where the product becomes less sustainable during its lifespan due to inefficient, slow or badly executed maintenance.

3.2 Software Metrics & Improvement

Simply looking at the social aspects of an organisation is not necessarily enough to determine whether or not 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. If you only look at the former, the system becomes a black box which says yes or no without giving recourse to the suppliers. Looking only at the former might lead you to choose a suboptimal product or create an impossibly complex survey. Questions about which data structure, algorithm or caching strategy are applied do not naturally lend themselves to being reduced to single optimal short-form questions, but are still critical to the performance of the product. 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. Take, for example, the web browser that you are using the read this article. Although browsing the web is the main functionality that it is being used for, it contains many more features such as theming, extensibility, screenshots, online saving of links, keeping track of videos being played, history, bookmarks, etc. These features are ones you may use occasionally, but do not constitute your typical usage of the browser. Therefore, these should not be taken into consideration 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 have been created that are precise enough for suppliers to offer to build the product. 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 would be the requirements: reading an email, sending an email and synchronising with the mail server. This *critical use-case* is a common set of features that should be available across all suppliers allowing implementation to be compared between each other and are indicative of the energy that would be expected to be used while using the software.

Another option would be to compare all code paths. The former is suboptimal since it would penalise implementations that offer more features or aim to minimise 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 to compare to a global best, the requirement still needs to be full-filled, nor to some unavailable optimal program.

3.4 Positive feedback cycle

These categories are not counted up together, but rather are contained in a virtuous cycle where each is necessary but not sufficient. Ignoring one of these aspects causing a problem to occur which might negate the benefits of the others.

By not including the foundational questions, the company might revert

Features	Strength	Drawback
Foundational + Health	Best practices from trusted partner	Suboptimal choice due hidden variables
Foundation + Metrics	Best product from trusted partner	No improvement suggestions
Health + Metrics	Best product following best practices	Sustainability may decrease over time

3.5 Cross-cutting question variants

3.6 Example Implementation of FISHER

For the use case of this study, a simple implementation using the principles of FISHER has been implemented. In this case, each section has an equal value for the final result.

3.7 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. The following criteria can be found in the Social Sustainability category:

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 the best 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.

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.

3.8 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 by giving concrete advice. They can also be seen as guidelines for non-experts. In this category the following software sustainability criteria can be found:

Programming language Some programming languages take longer, more memory and 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 much 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 information over the web requires a lot of 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 it be free software 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 and 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, allowing for the closer packing of software packages and higher 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 that some bugs do not 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.

3.9 Measurable Metrics

This category contains criteria specifically relating to the task that needs to be done. 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.

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.

4 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 setup 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.

4.1 Framework implementation

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

4.1.1 Tech stack

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

The backend for our experimental implementation is based on the article written by *Emen* on *Codevoweb.com* [1]. It's a multi-container docker system using a Nodejs application written in Typescript for the express API, a Mongo database for storage and a redis cache for session implementations. We opted for implementing authentication and authorization capabilities from the beginning to ensure the ability to extend this project in the future by transitioning to a fully integrated system that allows for comparison between different companies in the procurement process.

4.1.2 Implementation

For the tool, we directly re-used all the categories from the framework. In addition, almost all questions were re-used directly as well, with some exceptions in the Measurable Metrics category which we describe in the following sections. 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 we defined four possible answer types:

- 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.

Additionally, there is a fifth type available, a Likert scale, but this is not used within the framework or the tool. The mappings between the types directly relate to the types of questions in the framework. All concrete questions are defined as a boolean type, while conditional questions use the selector type. For the open questions like the metrics, the input type is used. For questions with multiple answers, the dropdown is used, such as for the programming languages. Questions with the selector type often also define a "Not Applicable" option, which eliminates the question

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

from the questionnaire results if selected. This is done to facilitate some way of configuration for projects where not every question is relevant due to the requirements of the project.

In the current proof of concept, each question and category has an equal weight. Also, each question's score is either zero or one for discrete questions or a fraction between zero and one. 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. This is the part that gives extra information about the question and helps to interpret the answers.

The scores are shown per category as a percentage, which allows for an easier comparison. The 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.

4.2 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 metric answers. For this reason, all questions from the Measurable Metric category have been changed from an open question to a yes or no question. Instead of asking for the measured values of the system, the question is changed to ask if such metrics are measured at all. In this way, the questions can still be used although in a different way compared to the framework. The main use for this system is that a client can quickly get an indication of the sustainability of a company or a company can get an indication of their own performance.

4.3 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 form 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.

In comparison to the standalone system, there is no change in the questions from the Measurable Metrics category, as in this system it is possible to directly compare companies and their provided metrics. In this way, the questions can be scored relative to the other companies and do not require a predefined score.

5 Future Work

5.1 Example Framework

The example framework could be extended on three main points. The first point is project-specific criteria. Each application type, e.g. desktop or mobile, should 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 green software procurement. The second point is 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 relatively equivalent projects, giving more insights into what a company can improve upon. The final point is physical software product criteria. Some software products are distributed using physical products, e.g. CDs. So, having specific criteria for these physical products is also needed.

5.2 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 allows for adding projects and filling out the form for a specific company in a project. 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 form 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.
- Form 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 run the front and back-end.

6 Conclusion

There has been little research done in the field of green software procurement. In this paper, we introduce a new framework, FISHER, which serves as a guideline for green 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 questions that are related to the sustainability domain of the category. These questions 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 on short and long-term sustainability aspects. Based on our theoretical framework we develop 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. With this framework, we wish

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

to assist with green 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.

[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.

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

[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.

[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.

[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.

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 on 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?*