An Exploration of Sustainability Thinking in Research Software Engineering

Timo Kehrer*, Birgit Penzenstadler†
*Humboldt-Universität zu Berlin, Germany
timo.kehrer@informatik.hu-berlin.de
†California State University, Long Beach, USA
birgit.penzenstadler@csulb.edu

***Abstract*—Research software is challenged by comparatively short educational cycles and intermediate academic funding cycles versus long-term research progress. Typical BSc, MSc and Phd projects, which contribute significantly to the development of research software, have a typical duration of 3 months to 4 years and while they ideally build on previous work, these works often turn into components that, after graduation, are shelved to die. This paper shows an avenue for research which can be characterized as rethinking the development of research software from a sustainability perspective. In order to get clarity on the underlying values and to extract exemplary sustainability requirements, we instantiate a sustainability reference model for the case example of two dedicated research software systems from the field of Model-Driven Engineering. This provides discussion grounds for a wider exploration of sustainable research software.**

I. Introduction

Science increasingly relies on different kinds of research software used for various purposes. We may distinguish two major categories of research software. In the natural, economic, social and life sciences, for instance, scientific discoveries are often data-driven, which in turn requires adequate software to generate, compute, analyze, and visualize the respective research data [18]. In the engineering sciences, at least in those which heavily rely on computer-aided analysis and design principles, empirical validation of new methods and techniques requires prototypical implementations and tools. These tools, being developed for the sake of gaining scientific insights, represent another kind of important research software. In particular, tools developed by software engineering researchers fall into the latter category of research software [35]. Some of the characteristics that make research software stand apart from other types of software systems are the funding structures (grants), academic recognition incentives (publications), and extremely distributed development. Such distributed development can include silos (individual research groups that do not communicate about common objectives) as well as geographically distributed developer teams.

Long-term research progress is particularly challenged by comparatively short educational cycles and intermediate academic funding periods. Considerable portions of research software are developed in terms of BSc, MSc and PhD projects which have a typical duration of just a few months or years. While they ideally build on previous work, these works often turn into components that, after graduation, are shelved to die. As a consequence, research results are often irreproducible, design knowledge gets lost quickly, and the development of research software becomes highly inefficient due to the repeated re-implementation of legacy software components. In addition to the lack of re-use and continuous evolution, some research software has a significant potential for broader application, which expands the initial use in an individual research project—a potential which is currently not fully explored.

Expanded usage scenarios and the phenomenon of rapid research software aging invoke a need for user-oriented evolution, maintenance, emulation, dissemination, and archiving of the software in question. Such kind of service leads to (technical) sustainability of research software and has to be enabled by designated technical and organizational infrastructures that supersede traditional versioning and documentation [12]. However, beyond quality assurance and user-driven evolution, sustainability concerns with regard to research software are not limited to the technical dimension but also include a **social** (user and developer communities), an **economic** (ROI), an **environmental** (in terms of impacts) [31], and an **individual** dimension (life-long learning).

Consequently, for research software to progress more towards sustainability, we need to explore the relation of (specific kinds of) research software to the other dimensions of sustainability in further detail. We aim at answering three research questions:

1) Was does sustainability mean for research software?
2) Which sustainability objectives can be identified for research software?
3) What are the ways in which we could progress towards achieving those objectives?

Research Design: We conduct an analysis of the sustainability dimensions for research software according to the generic sustainability goal model [31]. In brief, the dimensions can be thought of as viewpoints or lenses (which means that contents can overlap), and are denoted as individual, social, economic, environmental, and technical.

Contribution: Using two concrete research tools from the field of Model-Driven Engineering (MDE) as a case example, we illustrate the analysis of this specific kind of research software under the lens of the five dimensions of sustainability. From this analysis, we discuss which of the proposed measures can be generalized from the specific tools to other research

software, and where other kinds of research software may lead to further sustainability requirements which are of minor importance for the case example used in the paper at hand.

Impact: This contribution could lay the foundation for a research agenda for sustainable research software.

The remainder of the paper is structured as follows: Section II presents the background and related work, Section III presents the instantiation of the model and analysis, Section IV discusses limitations thereof, and Section V concludes the paper with an outlook on next steps.

II. Background & Related Work

In Section II-A, we review the work related to the paper at hand, starting with a high-level overview of development methods for (not necessarily scientific) long-living software systems, and subsequently narrowing our scope to sustainable research software. Thereafter, in Section II-B we briefly recall the general methodology for requirements engineering for sustainability applied in this paper. Finally, in Section II-C we give an overview of the scientific domain and the kind of research software serving as a case example in this paper, i.e., for which we instantiate this methodology.

A. Related Work: Sustainable Research Software

Many software engineering sub-disciplines directly or indirectly contribute to the goal of mitigating software aging by supporting the continuous evolution of long-living software systems with respect to ever-changing requirements and technical platforms. For example, the ICSME[1], one of the premier forums for researchers and practitioners to present and discuss the most recent innovations, trends, experiences, and challenges in software maintenance and evolution, compiles more than 20 topics of interest around maintenance and evolution. Many of these topics are also picked up by other major software engineering conferences, such as ICSE[2], ASE[3] and ESEC/FSE[4]. However, virtually all of these topics — such as change and defect management, clone management, reverse engineering and re-engineering, quality assurance, to mention just a few of them — clearly target the technical dimension of sustainability, while the other dimensions are mostly neglected. The same applies to most larger recent research initiatives. For example, the research structure of the German priority program "Design for Future – Managed Software Evolution" [16], although having a determined focus on long-living software systems, aims at technical improvements such as knowledge carrying software, methods and processes as well as platforms and environments for software evolution.

While there have been several works in the area of sustainable software in the sense of technical (and sometimes economic) sustainability, e.g. [13], [26], in the paper at hand we focus on research software as defined in the introduction. Other works focused mainly on green software engineering in a limited scope of the environmental dimension, usually dominated by energy-efficiency (see, e.g., [28], [29]).

In contrast, the series of Workshops on Sustainable Software for Science: Practice and Experiences (WSSSPE) focuses specifically on research software. So far, there have been held four instances. The Fourth Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE4) was co-located in Manchester with the First Conference of Research Software Engineers (RSE Conference). WSSSPE3 was held on 28–29 September 2015 in Boulder, Colorado, USA. Previous events in the WSSSPE series are WSSSPE1, held in conjunction with SC13 (the Intl. Conference for High Performance Computing, Networking and Analysis); WSSSPE1.1, a focused workshop organized jointly with the SciPy conference[5]; WSSSPE2, held in conjunction with SC14; and WSSSPE2.1, a focused workshop organized again jointly with SciPy. The WSSSPE1 workshop engaged the broad scientific community to identify challenges and best practices in areas relevant to sustainable scientific software. WSSSPE2 invited the community to propose and discuss specific mechanisms to move towards an imagined future practice of software development and usage in science and engineering. WSSSPE3 organized self-directed teams that collaborated prior to and during the workshop to create vision documents, proposals, papers, and action plans. They produced a JORS report to document the vision sessions and results of team discussions [21].

One of the initiatives coming out of WSSSPE is the conceptualization of a US Research Software Sustainability Center (URSSI) funded by the NSF [9]. They identified three primary classes of concern are pervasive across research software: functioning of the individual and team, functioning of the research software, and functioning of the research field itself. The goal of the current conceptualization project is to create a roadmap for a URSSI to minimize or at least decrease these types of concerns [9].

Finally, in 2016, there was a Dagstuhl Perspectives Workshop on Engineering Scientific Software that proposed a set of pledges for citations, careers, and development and use of scientific software—and furthermore proposed six directions for future research: Quantifying the availability of scientific software, facilitating software discovery within and across disciplines, sustainability of software experimentation, software engineering tools improving productivity by tailoring to intent and skill re-tooling the bibliographic software toolchain for software citation, and analysis of scientific software ecosystem metadata [2].

B. Background: Requirements Engineering for Sustainability

Requirements are the key leverage point for practitioners who want to develop sustainable software-intensive systems [5]. In order to develop such systems, we need awareness (by education), guidance (e.g., by training), and creativity (to

[1] 34th International Conference on Software Maintenance and Evolution (ICSME) https://icsme2018.github.io/

[2] 40th International Conference on Software Engineering (ICSE) https://www.icse2018.org/

[3] 33rd International Conference on Automated Software Engineering (ASE) http://www.ase2018.com/

[4] 26th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) https://conf.researchr.org/home/fse-2018

[5] https://conference.scipy.org

find better solutions). We use the term RE4S as defined in previous work [30]: Requirements Engineering for Sustainability (RE4S) denotes "the concept of using requirements engineering and sustainable development techniques to improve the environmental, social, and economic sustainability of software systems and their direct and indirect effects on the surrounding business and operational context" [30].

The RE4S approach uses an artifact model, guiding questions, checklists, and reference models to elaborate the requirements for a system under development. The entire approach is described in detail in [30] and example specifications have been provided in [32], [11]. The approach uses five sustainability dimensions, for which we refer to the definition in [5]:

- The **individual dimension** covers individual freedom and agency (the ability to act in an environment), human dignity, and fulfillment. It includes individuals' ability to thrive, exercise their rights, and develop freely.
- The **social dimension** covers relationships between individuals and groups. For example, it covers the structures of mutual trust and communication in a social system and the balance between conflicting interests.
- The **economic dimension** covers financial aspects and business value. It includes capital growth and liquidity, investment questions, and financial operations.
- The **technical dimension** covers the ability to maintain and evolve artificial systems (such as software) over time. It refers to maintenance and evolution, resilience, and the ease of system transitions.
- The **environmental dimension** covers the use and stewardship of natural resources. It includes questions ranging from immediate waste production and energy consumption to the balance of local ecosystems and climate change concerns.

Some approaches for modeling and reasoning about concepts supporting software sustainability in its various dimensions have been proposed in the literature, e.g., in [7], [31]. In the contribution at hand, we focus specifically on the goal modeling aspect. In [31], Penzenstadler and Femmer present a reference model for sustainability that decomposes sustainability into these five dimensions. The model provides activities and relates them to values they support and assessable indicators. It is intended to serve as a reference model for a process engineer who instantiates the model for a software development company or for a requirements engineer who instantiates it for a specific system under development.

A subset of the meta-model of this generic sustainability model, originally introduced in in [31], is shown in Figure 1. It is comprised by the types *Goal*, *Dimension*, *Value*, *Indicator*, and *Activity*: A <Goal> may be approached from several dimensions representing different aspects of the overall goal. A <Dimension> is a specific viewpoint, represented by a set of values that express the abstract objectives of the dimension. The dimensions may influence each other. A <Value> is a rationale that is rooted in itself, while an <Objective> explains what a value means for a specific instance, which may be approximated by indicators. An <Indicator> is a qualitative or

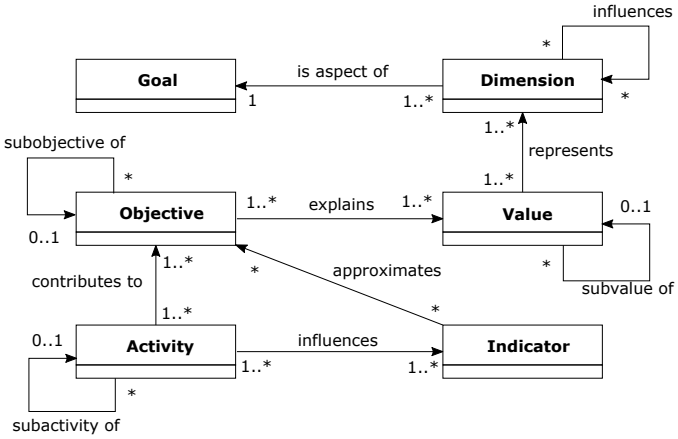

Figure 1. Subset of the meta-model of the generic sustainability model used in the paper at hand, adopted with slight extensions from [31].

quantitative metric and is related to a value. A <Regulation> is an optional element that affects a value. An <Activity> is a means to support a set of objectives and influences a subset of the indicators approximating the supported objectives. Values, activities, and objectives may be (recursively) decomposed into sub-values, sub-activities and sub-objectives, respectively. Please note that the meta-model shown in Figure 1 abstains from presenting some of the details introduced in [31] which are not used in the remainder of this paper, e.g., the usage of regulations as an optional element that affects a value. On the contrary, the paper at hand extends the original meta-model in [31] by introducing objectives as concrete refinements of abstract values.

Building on the five dimensions, the notion of further framing sustainability as a property of software quality was discussed in Lago et al. [27]. One of the key insights in that work was that sustainability requirements and concerns will increase system scope, requiring extended analysis during requirements engineering. This notion was also observed for the investigation of sustainability and application of the generic goal model to research software presented in the paper at hand.

C. Background: Scientific Domain of Model-Driven Engineering

Model-Driven Engineering (MDE) [6] is a software development paradigm which aims to ease the transition between informally sketched designs and implementations by supporting high-level yet formal (domain-specific) models as a starting point for automation, with the ultimate goal of increasing productivity and quality of developments. The increase of productivity, in turn, highly depends on the quality of the provided tool environment, which has to be customized to (domain-specific) modeling languages, development processes, and project-specific or individual preferences. Thus, large portions of MDE research are centered around the development of MDE tools; the kind of research software which we use as a case example of a scientific domain of interest in the remainder of this paper. More specifically, we consider two concrete tools within this domain which are still under research and development, known as Henshin and SiLift. The first author

of the paper at hand is actively involved in the development of both of these tools.

Exemplary MDE research tool 1: Henshin. Henshin [4], [37] is a model transformation framework based on the foundations of algebraic graph transformation [14]. It provides a high-level model transformation language, including both declarative and imperative language constructs, as well as an interpreter for model transformation execution and simulation. Furthermore, it supports various techniques for statically analyzing model transformation systems, such as state space exploration facilities and a prototypical implementation of a formal calculus known as critical pair analysis. Using a simple form of the well-known dining philosophers example, Figure 2 gives an intuition of rule-based model transformation specifications in Henshin and its integrated state space generation and analysis facilities.

Since model transformations are often considered as the "heart and soul" of MDE [34], Henshin is a representative kind of research software in our scientific domain of interest.

Exemplary MDE research tool 2: SiLift. SiLift [23] is a generic yet highly adaptable model management framework and tool suite which can be instantiated for a wide range of model management tasks supporting model evolution, including services for model versioning and model-driven software product-line engineering. The core services provided by the framework are generally known as model differencing, patching and merging, all of which work on a structural, graph-based representation of the underlying models and use sets of language-specific change operations in order to handle model modifications. For example, using the SiLift framework, a model differencing service may be instantiated which is capable of recognizing high-level change operations such as language-specific refactorings between two versions of a model. A screenshot of the graphical user interface of a SiLift integration into the UML modeling tool Papyrus[6] is shown in Figure 3.

With models becoming primary development artifacts in MDE which are subject to continuous evolution, SiLift represents another class of highly relevant research software, namely model management tools in MDE.

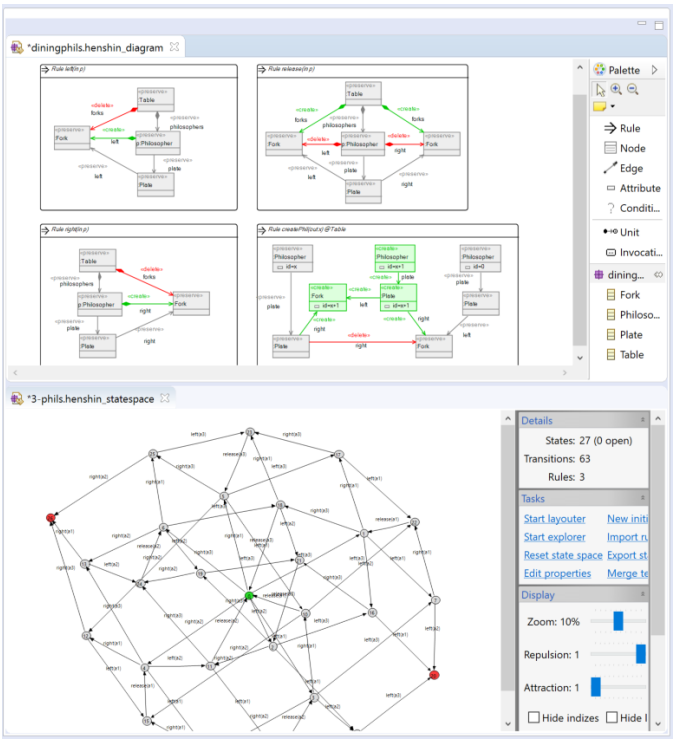

Figure 2. Illustration of the Henshin model transformation language and tool using a simple form of the well-known dining philosophers example: Rule-based model transformation specifications (top) and integrated state space generation and analysis facilities (bottom).

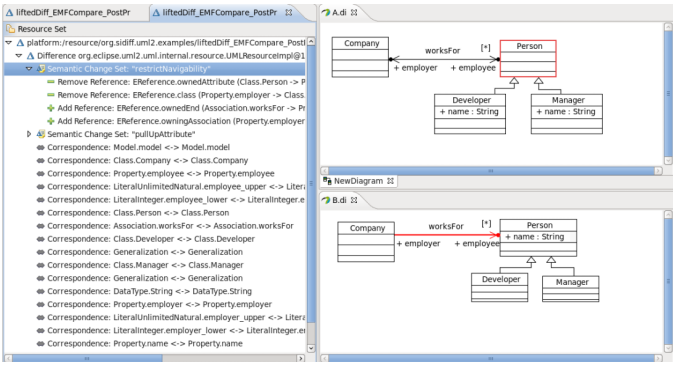

Figure 3. Screenshot of the graphical user interface of an integration of the SiLift model differencing service into the UML modeling tool Papyrus: Visualization of the difference calculated between two versions of a simple UML class diagram.

III. Instantiation of the Generic Sustainability Model for Research Software

In this section, we describe the instantiation of the generic sustainability model for research software. For the purpose of clarity and simplicity, we did not include all values, objectives and activities that could be discussed, but an exemplary selection that provides an overview of the most interesting elements (see Figure 4 below). In particular, we leave out those aspects which are not necessarily distinguishing for research software. Please note that we do not claim our model to be complete, but it is rather meant to provide a basis for further discussion and elaboration. Moreover, we do not claim generality, but we deliberately consider research software in the domain of Model-Driven Engineering. More specifically, we reflect on our experience with concrete research tools where we are actively involved in the development, namely Henshin and SiLift (see Section II-C). We performed the analysis phase described in [31], where the generic goal model is tailored to that context. This formalizes what sustainability really means for these research tools and, to some extent, other research software as discussed further below. That means that:

1) the values are instantiated to specific objectives, then
2) activities need to be defined to implement the goals, and
3) indicators assess how well the objectives have been achieved.

In the following, we discuss the represented elements per dimension for the instance of the case example Henshin, in

[6] https://www.eclipse.org/papyrus

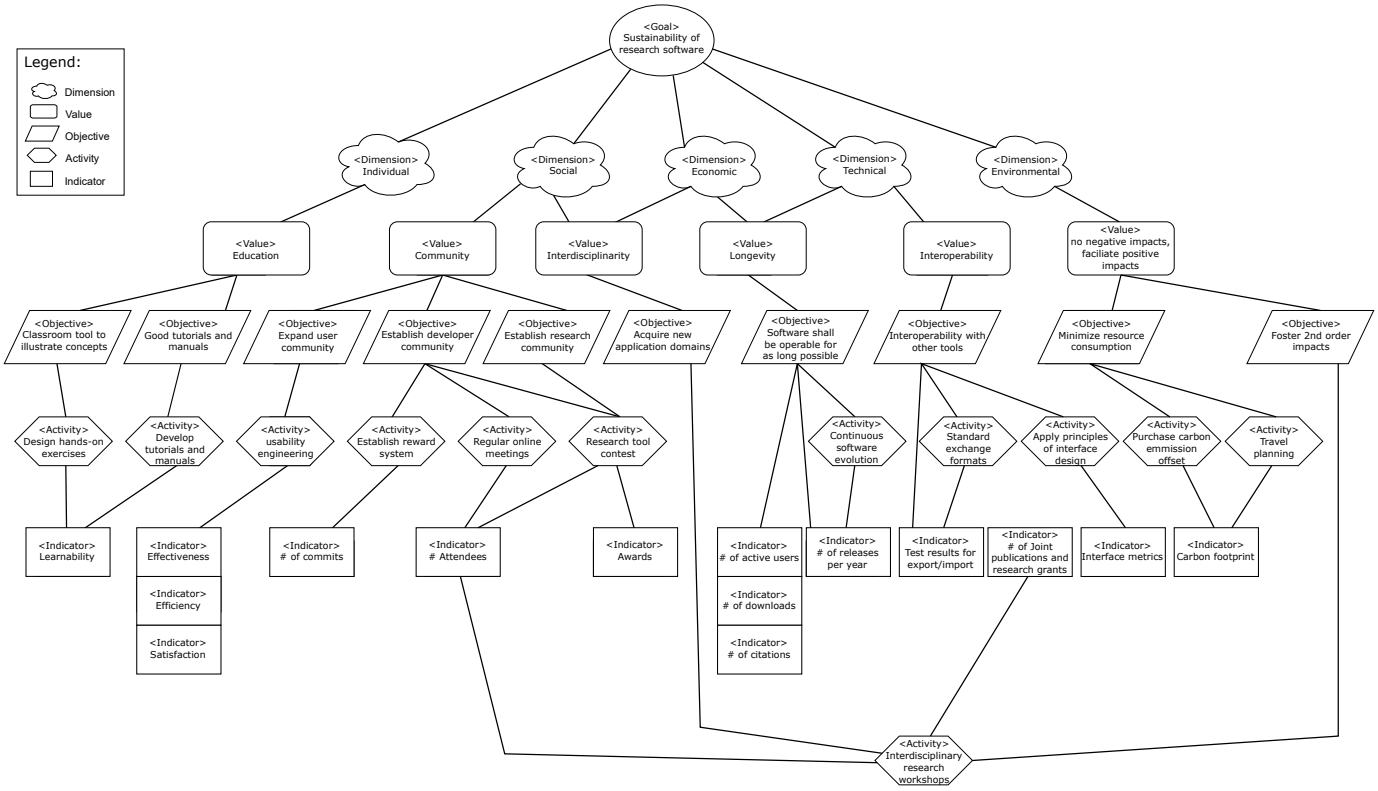

Figure 4. Excerpt of the sustainability model instance for research software, focusing on research software developed in the field of Model-Driven Engineering and reflecting on experience with two concrete research tools; Henshin and SiLift.

the order depicted in Figure 4. Please note that, while we are describing the model per dimension, there is no strict hierarchy, and the cross-references are better visualized in the diagram.

A. Individual Dimension

In addition to its primary purpose of generating new scientific insights, research software is often used for **education** in the classroom, for example, to illustrate software engineering concepts. As they are implemented in a (to the researcher / educator) well-known tool that students may be allowed to work with, they provide for an excellent learning environment. A prerequisite for such educational use is the availability of good tutorials and manuals in order for the students to be able to easily pick up the usage of the tool. Activities that can support this objective are tutorials on websites and wikis as well as in videos or forums. We use both Henshin and SiLift for education in the classroom, providing a set of hands-on exercises as starting points for the students. Indicators of success are any means that help assess the learnability of the tool, i.e., how well students perform for concrete tasks which are to be solved with the help of the tool. Concrete examples which we have used in the past are implementing dedicated model transformations in Henshin, or using the model differencing facilities of SiLift to answer questions about the historical evolution of a model.

B. Social Dimension

In the social dimension, the sustainability of research software can be significantly increased by building larger communities within the scientific domain of interest.

The **developer community** comprises active developers contributing to the research software as well as (typically more senior) researchers participating in strategic discussions. According to our experience, an activity which works quite well to form a developer community and serves as an instrument for motivation is to adopt open-source principles and ideas. The Henshin project, for instance, is hosted as an official Eclipse project[7]. This not only ties the Henshin development process to some of the lightweight quality assurance principles which must be implemented by any official Eclipse project (such as unit testing and code reviews), it implicitly introduces a reward system through which developers may increase their personal reputation. For instance, the amount of commits per developer is now visible to the general public. We found this to have a positive impact on students' motivation, particularly due to the visibility of the Eclipse project.

The loose collaboration via the open-source infrastructure may be supported, e.g., by regular online and physical meetings. In the Henshin project, for instance, a monthly online meeting of about one to two hours serves as a discussion forum for efficiently discussing major design decisions when a new feature is realized. The first Henshin developer day was held as a physical meeting at the Technical University of Darmstadt

[7] https://www.eclipse.org/henshin

in 2017, and the event shall take place in an annual fashion with the aim of fostering strategic discussions.

More often than not, the larger **research community** of a scientific domain comes up with a variety of different tools which are being developed for the same or at least similar purpose. In the MDE community, for instance, Henshin is by far not the only model transformation tool. Actually, a whole bunch of tools has been developed for the same purpose, yet following different transformation concepts and paradigms. Since all of these techniques and tools have their particular strengths and weaknesses, the research question arises which model transformation approach may be useful for which application domains and kinds of model transformation scenarios. A suitable activity to, at least gradually, approach possible answers to this question and to foster the collaboration within the larger research community is the regular organization of tool contests. The Model Transformation Tool Contest (TTC)[8], for instance, has been established in 2007 and aims to evaluate and compare quality attributes of model transformation tools such as expressiveness, usability and performance along selected challenging case studies (for example, case descriptions of the TTC 2017 may be found in [3], [15], [20]).

The tool(s) that perform best in a particular case study receive an award at the physical TTC meeting, which serves as an indicator for the suitability in a certain transformation scenario. The ultimate goal is to advance research on model transformation by evaluating, improving and integrating existing transformation tools, and by identifying open problems. Similar goals were pursued by the Intl. Workshop Series on Comparison and Versioning of Software Models (CVSM)[9], which initiated the development of a community benchmark set serving as a basis for evaluating and comparing tools for model differencing and merging.

Finally, the **user community** refers to those researchers of a larger scientific domain which do not actively contribute to the development of the research software (or any alternative tool for the same purpose), but who actively use the tool for their own research. Extending this user community is a highly desirable objective, e.g., for the sake of collecting valuable feedback in order to validate and improve the research software in question. A basic prerequisite for this is that the research software is easy to learn and to use which, in addition to accompanying tutorials, should be supported by applying basic principles from the field of usability engineering.

In the Henshin project, for instance, a user study was conducted to learn about tool users' preferences with respect to a textual syntax of the Henshin transformation language [37]. In addition, the usability may be significantly increased by providing meta-tools for adapting a tool to a specific use case and context. As an example, consider again the Henshin transformation language, in which model transformation rules provide the basic building blocks of any model transformation. The definition of such rules can be supported by learning transformation rules from user-provided examples [22], or by generating basic rules from a given meta-model of a modeling language [25], [33] or UML profile [24]. Classical usability metrics such as the effectiveness, efficiency and user satisfaction for a set of dedicated tasks are indicators affected by these activities.

The same meta-tools can also be used to adapt the generic services provided by the SiLift framework to a given (domain-specific) modeling language. Moreover, SiLift is highly experimental in nature, exposing a huge configuration space for tailoring details via plenty of configuration parameters, e.g., for the selection of proper algorithms and for further tuning these algorithms through numerical parameters. To assist tool users in choosing a suitable configuration, we applied principles of software product-line engineering which enable a high-level and easy to use configuration process. However, there is still much room for improvement, e.g., through automatically learning optimal configurations for dedicated usage scenarios of the tool.

C. Economic Dimension

Besides establishing native developer, researcher and user communities within a scientific domain of interest, it is worth exploring the potential **interdisciplinary** usage scenarios of a research software in order to increase both social and economic sustainability. For instance, to enable heuristic search over models expressed in domain-specific languages, Henshin has been successfully utilized to encode high-level model modifications such as genetic mutation operators [38]. The overall goal of this search-based model engineering paradigm is to lower the access barrier to applying the current state of the art in optimization and search to specific problems. This application still requires substantial expertise in optimization research since problem encodings are not usually naturally derived from domain knowledge. Search and optimization can be applied to a large number of (combinatorial) problems, e.g., in the larger context of search-based software engineering [17] or even beyond. To acquire new application domains and explore such potential research synergies, a supporting activity is to organize interdisciplinary workshops serving as a platform for fostering collaborations. An example of this is the HU-KCL Workshop on Search-Based Model Engineering[10] co-organized by representatives from HU Berlin and King's College London, to which we invited researchers from geo-sciences, bio-informatics, and others. The number of joint research projects and publications may serve as an indicator for this value.

Along a similar vein, SiLift has been successfully used in a number of research collaborations in order to solve concrete problems beyond its traditional application domain of model versioning. For example, an approach and supporting tool for reasoning about software product-line evolution using differences in feature models has been developed in [8]. Beyond usual syntactic model differencing, a specific contribution of this collaboration is a facility for detecting complex editing operations on feature models. Their semantic impact on the set of valid feature configurations can be classified semantically as

[8] https://www.transformation-tool-contest.eu
[9] http://pi.informatik.uni-siegen.de/CVSM2014
[10] https://www.informatik.hu-berlin.de/de/forschung/gebiete/mse/HU-KCL-Workshop

refactoring, generalization or specialization of a feature model. More generally, the SiLift tool turned out to be a cross-cutting technology which is widely used within the already mentioned research initiative on long-living software systems [16], [39].

Longevity refers to the objective that a specific research software shall be operable as long as possible. From an economic perspective, the main asset which needs to be maintained is the added value contained by research software components developed in the context of BSc, MSc and PhD theses. From a technical point of view, longevity needs to be assisted by traditional software engineering methods and techniques dealing with long-term software evolution in the context of changing technologies, organizational constraints and requirements. Since there is plenty of related work in the technical dimension (see Section II-A), this is summarized under the supporting activity referred to as continuous software evolution in Figure 4 but not further elaborated in the paper at hand. A rough indicator for continuous software evolution is the number of releases per year.

D. Technical Dimension

Research results are seldom produced using just a single tool. In the MDE context, for instance, development environments typically consist of complex tool chains comprising sets of heterogeneous tools which are supposed to heavily interact with each other. To perform larger experiments and evaluate certain tooling aspects in realistic case studies, such an integration into an existing MDE environment is often inevitable, and thus **interoperability** with other tools is an important value in our scientific domain of interest. There are well-known activities for supporting this goal. For example, exchange format standardization and design principles for interface design, which may be assessed, e.g., through interoperability testing and by using interface metrics which measure, e.g., the complexity and usage cohesion of interfaces [1]. However, these principles are often neglected by research software engineers and need to be further emphasized to finally achieve technical sustainability. In the historical evolution of both SiLift and Henshin, for instance, a major design decision was to adopt the model representation of the Eclipse Modeling Framework (EMF) [36], which emerged as the de-facto standard for representing models within the MDE research community. This design decision had a significant impact and enabled us to validate the tools in a much broader context.

E. Environmental Dimension

The generic value of any research software is that it has no (or at least minimizes) negative and facilitates positive environmental impacts.

Even if positive environmental impacts are hard to foresee for many kinds of research software, e.g., for experimental prototypes in the engineering sciences, such research software may lead to 2nd order impacts through acquiring new application domains. This is particularly the case if some research software can be applied in other scientific domains in which new research results may have a direct positive environmental impact. Consider again the potential interdisciplinary research synergies arising from the search-based model engineering paradigm introduced in Section III-C. In bio-informatics and various natural sciences, for instance, computational methods are used to characterize scientific objects — such as gene sequences or physical materials structures — using a mixture of software tools. A major problem is to find a suitable combination of (configurations of) these tools to perform computations that meet certain constraints and quality requirements, which in turn may be supported by model-based optimization methods.

In addition, a rather traditional way to minimize negative impacts, one concrete exemplary objective is to reduce energy consumption, which we consider here from a very broad perspective. On the one hand, we may reduce the computational resources consumed by the resource software itself. However, choosing an appropriate technical paradigm for resource-aware programming heavily depends on the kind of developed research software and the hardware architecture it is supposed to run on, thus computational resource efficiency will be not considered here any further. On the other hand, a much more generalizable activity is to properly plan the physical meetings of research software developers and collaborating researchers within a scientific domain, with the overall goal of reducing traveling efforts. One obvious way to avoid traveling is to prefer online discussions over physical meetings where possible. If online meetings are highly inadequate, e.g., for the kinds of physical meetings briefly discussed in Section III-B, such supporting meetings may be efficiently organized in co-location with other major events. The transformation tool contest (see Section III-B), for instance, is organized within the STAF conference series, which hosts the most specific scientific conferences being of interest for any researcher working in the domain of model transformations. When traveling cannot be avoided, a supporting activity is to buy carbon offsets along with flight tickets in order to reduce individual carbon footprints.

IV. Limitations

In this section, we discuss the two major limitations of the model presented in the previous section in more detail.

A. Generalizability of the Model

As already mentioned, we do not claim generality for our model presented in Section III. Some of the elements discussed are highly specific for the research software under consideration, namely research tools in MDE in form of the Henshin model transformation tool suite as well as the model versioning and management framework SiLift. While we believe that the model is general enough to capture sustainability requirements from the same scientific domain, i.e., experimental software developed in the context of MDE, some aspects are hardly transferable to other scientific domains.

Consider, for instance, the computational sciences, where research software is often developed for a very specific purpose, e.g., dedicated data analysis workflows for earthquake and flood warning systems. On the one hand, some of the

values and objectives which we considered to be important for research software in the engineering sciences are of minor importance in the computational sciences. For example, the general cross-domain applicability of the research software manifests itself in objectives such as the expansion of the user community or the acquisition of new application domains. This is hardly an issue for a research tool that has been developed for such a specific purpose like earthquake prediction.

Moreover, since such research software is not a prototypical implementation or supporting vehicle for an engineering technique or method, it is not typically used as a classroom tool that would illustrate dedicated concepts of a certain domain. On the other hand, environmental impacts that arise from the usage of a system in its application domain are much more clearly identifiable [19].

B. Completeness of the Model

Just like for generalizability, we do not claim completeness of our model presented in Section III. While it covers a set of important sustainability requirements for research software developed in MDE, there are others which are missing in the current version of our model. This is just due to the fact that they have not yet been a major issue in our research software under consideration.

For example, research results achieved using a software tool shall be reproducible, even long after initial publication. This applies in particular to the data-driven sciences and distinguishes research software from many other kinds of software where users are only interested in the latest version comprising the most fancy features. While we have many ways of archiving software, they are not necessarily sufficient for describing everything required for reproducing empirical research results.

The most significant progress we have been able to detect for research software in terms of sustainability concerns is evident in the technical dimension [21]. However, also in this dimension there remain significant challenges, some of which may require entirely new software engineering methods. In fact, many advanced software engineering principles cannot be simply adopted for the scientific domain due to the inherent differences between scientific analysis and its counterparts in other traditional fields. For instance, the usage of formal methods, as developed in the context of safety- and mission critical systems, require up-front formal requirements specifications. These are typically not available in computational science where requirements often emerge in an interactive and iterative discovery process [10]. For similar reasons, advances in software testing are hardly applicable due to the lack of test oracles, which ultimately result from the explorative nature of scientific data analysis. These examples show that research software engineering in these domains is not merely the application of established software engineering techniques to scientific data analysis problems. Instead, such problems actually generate new research challenges for software engineering.

V. Conclusion and Future Work

In this section, we summarize the insights of the paper at hand as well as give an outlook on future work.

A. Summary

The paper at hand instantiated the generic sustainability goal reference model [31] for research software in order to rethink the development of research software from a sustainability perspective. Specifically, our exploration was guided by the following questions:

1) *Was does sustainability mean for research software?* We instantiated the generic sustainability model with specific values and objectives for research software and find that there is much potential future work to help transition research software towards a more sustainable development process.
2) *What sustainability objectives can be identified for research software?* We identified objectives across all five sustainability dimensions, nine of which are illustrated in Figure 4. The exploration of the five dimensions helped in expanding the scope for an analysis of the sustainability of research software.
3) *What are the ways in which we could progress towards achieving those objectives?* The objectives can be implemented via activities and assessed via indicators as exemplarily shown in Figure 4. Next steps are to implement some of the activities for the research tools Henshin and SiLift considered in this paper.

At the current state of our research, we neither claim generalizability and completeness of our sustainability goal model, nor do we claim the general applicability of the overall method used in this paper. While it helped us to formalize several aspects of what sustainability really means for the two research tools of our case example, the analysis was carried out in a very restricted setting involving just a few stakeholders. It may therefore not easily scale to larger projects or be transferred to other scientific domains. However, we are convinced that the method applied is a good starting point within individual groups of work, specific research communities, etc. for research software to progress further towards sustainability. This would require a method for systematic application of and guidance for activities supporting the development of sustainable research software.

B. Future Work

There are two major directions for future work. First, for the considered kind of research software, namely experimental tools in Model-Driven Engineering, the next step is to enter the application & assessment phase of the method presented in [31]. That is, all the activities which have been identified during the analysis phase shall be systematically applied in the context of the Henshin development project. Their impact on the desired sustainability values shall be measured by the respective indicators. Such monitoring and supervision will provide a natural validation of our approach as well as the necessary feedback to further refine our sustainability model

presented in this paper. Second, following our discussion in Section IV, we are interested in to what extent our sustainability model can be generalized to other kinds of experimental MDE tools and, from a more long-term perspective, to other kinds of research software from entirely different scientific domains.

References

[1] Hani Abdeen, Houari Sahraoui, and Osama Shata. How we design interfaces, and how to assess it. In *29th IEEE International Conference on Software Maintenance*, pages 80–89. IEEE, 2013.

[2] Alice Allen, Cecilia Aragon, Christoph Becker, Jeffrey Carver, Andrei Chis, Benoit Combemale, Mike Croucher, Kevin Crowston, Daniel Garijo, Ashish Gehani, et al. Engineering academic software (dagstuhl perspectives workshop 16252). In *Dagstuhl Manifestos*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[3] Anthony Anjorin, Thomas Buchmann, and Bernhard Westfechtel. The families to persons case. In Antonio García-Domínguez, Georg Hinkel, and Filip Krikava, editors, *Proceedings of the 10th Transformation Tool Contest (TTC 2017), co-located with the 2017 Software Technologies: Applications and Foundations (STAF 2017), Marburg, Germany, July 21, 2017.*, volume 2026 of *CEUR Workshop Proceedings*, pages 27–34. CEUR-WS.org, 2017.

[4] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: advanced concepts and tools for in-place emf model transformations. In *International Conference on Model Driven Engineering Languages and Systems*, pages 121–135. Springer, 2010.

[5] Christoph Becker, Stefanie Betz, Ruzanna Chitchyan, Leticia Duboc, Steve M Easterbrook, Birgit Penzenstadler, Norbet Seyff, and Colin C Venters. Requirements: The key to sustainability. *IEEE Software*, 33(1):56–65, 2016.

[6] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

[7] Isabel Sofia Brito, José M Conejero, Ana Moreira, and Joao Araújo. A concern-oriented sustainability approach. In *12th International Conference on Research Challenges in Information Science*. IEEE, 2018.

[8] Johannes Bürdek, Timo Kehrer, Malte Lochau, Dennis Reuling, Udo Kelter, and Andy Schürr. Reasoning about product-line evolution using complex feature model differences. *Automated Software Engineering*, 23(4):687–733, 2016.

[9] Jeffrey C Carver, Sandra Gesing, Daniel S Katz, Karthik Ram, and Nicholas Weber. Conceptualization of a us research software sustainability institute (urssi). *Computing in Science & Engineering*, 20(3):4–9, 2018.

[10] Jeffrey C Carver, Richard P Kendall, Susan E Squires, and Douglass E Post. Software development environments for scientific and engineering software: A series of case studies. In *Proceedings of the 29th international conference on Software Engineering*, pages 550–559. IEEE Computer Society, 2007.

[11] Ruzanna Chitchyan, Stefanie Betz, Leticia Duboc, Birgit Penzenstadler, Steve Easterbrook, Christophe Ponsard, and Colin Venters. Evidencing sustainability design through examples. In *Intl. Workshop on Requirements Engineering for Sustainable Systems 2015*, 2015.

[12] German Research Foundation (DFG). Call for proposals: Research software sustainability. http://www.dfg.de/download/pdf/foerderung/programme/lis/161026_dfg_call_proposal_software_en.pdf, 2016.

[13] Zoya Durdik, Benjamin Klatt, Heiko Koziolek, Klaus Krogmann, Johannes Stammel, and Roland Weiss. Sustainability guidelines for long-living software systems. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 517–526. IEEE, 2012.

[14] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

[15] Sinem Getir, Duc Anh Vu, Francois Peverali, Daniel Strüber, and Timo Kehrer. State elimination as model transformation problem. In Antonio García-Domínguez, Georg Hinkel, and Filip Krikava, editors, *Proceedings of the 10th Transformation Tool Contest (TTC 2017), co-located with the 2017 Software Technologies: Applications and Foundations (STAF 2017), Marburg, Germany, July 21, 2017.*, volume 2026 of *CEUR Workshop Proceedings*, pages 65–73. CEUR-WS.org, 2017.

[16] Ursula Goltz, Ralf H Reussner, Michael Goedicke, Wilhelm Hasselbring, Lukas Märtin, and Birgit Vogel-Heuser. Design for future: managed software evolution. *Computer Science-Research and Development*, 30(3-4):321–331, 2015.

[17] Mark Harman and Bryan F Jones. Search-based software engineering. *Information and software Technology*, 43(14):833–839, 2001.

[18] Tony Hey, Stewart Tansley, Kristin M Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA, 2009.

[19] Lorenz M Hilty and Bernard Aebischer. Ict for sustainability: An emerging research field. In *ICT Innovations for Sustainability*, pages 3–36. Springer, 2015.

[20] Georg Hinkel. The TTC 2017 outage system case for incremental model views. In Antonio García-Domínguez, Georg Hinkel, and Filip Krikava, editors, *Proceedings of the 10th Transformation Tool Contest (TTC 2017), co-located with the 2017 Software Technologies: Applications and Foundations (STAF 2017), Marburg, Germany, July 21, 2017.*, volume 2026 of *CEUR Workshop Proceedings*, pages 3–12. CEUR-WS.org, 2017.

[21] Daniel S Katz, Sou-Cheng T Choi, Kyle E Niemeyer, James Hetherington, Frank Löffler, Dan Gunter, Ray Idaszak, Steven R Brandt, Mark A Miller, Sandra Gessing, et al. Report on the third workshop on sustainable software for science: Practice and experiences (wssspe3). *Journal of Open Research Software*, 4(1):37, 2016.

[22] Timo Kehrer, Abdullah Alshanqiti, and Reiko Heckel. Automatic inference of rule-based specifications of complex in-place model transformations. In *International Conference on Theory and Practice of Model Transformations*, pages 92–107. Springer, 2017.

[23] Timo Kehrer, Udo Kelter, Manuel Ohrndorf, and Tim Sollbach. Understanding model evolution through semantically lifting model differences with silift. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 638–641. IEEE, 2012.

[24] Timo Kehrer, Michaela Rindt, Pit Pietsch, and Udo Kelter. Generating edit operations for profiled UML models. In *Proceedings of the Workshop on Models and Evolution co-located with ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013), Miami, Florida, USA, October 1, 2013.*, volume 1090 of *CEUR Workshop Proceedings*, pages 30–39. CEUR-WS.org, 2013.

[25] Timo Kehrer, Gabriele Taentzer, Michaela Rindt, and Udo Kelter. Automatically deriving the specification of model editing operations from meta-models. In *International Conference on Theory and Practice of Model Transformations*, pages 173–188. Springer, 2016.

[26] Heiko Koziolek, Dominik Domis, Thomas Goldschmidt, and Philipp Vorst. Measuring architecture sustainability. *IEEE software*, 30(6):54–62, 2013.

[27] Patricia Lago, Sedef Akinli Koçak, Ivica Crnkovic, and Birgit Penzenstadler. Framing sustainability as a property of software quality. *Communications of the ACM*, 58(10):70–78, 2015.

[28] Sara S Mahmoud and Imtiaz Ahmad. A green model for sustainable software engineering. *International Journal of Software Engineering and Its Applications*, 7(4):55–74, 2013.

[29] Stefan Naumann, Markus Dick, Eva Kern, and Timo Johann. The greensoft model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1(4):294–304, 2011.

[30] Birgit Penzenstadler. Infusing green: Requirements engineering for green in and through software systems. In *Intl. Workshop on Requirements Engineering for Sustainable Systems@ RE*, pages 44–53, 2014.

[31] Birgit Penzenstadler and Henning Femmer. A generic model for sustainability with process-and product-specific instances. In *Proceedings of the 2013 workshop on Green in/by software engineering*, pages 3–8. ACM, 2013.

[32] Birgit Penzenstadler, Joseph Mehrabi, and Debra J Richardson. Supporting physicians by re4s: Evaluating requirements engineering for sustainability in the medical domain. In *Proceedings of the Fourth International Workshop on Green and Sustainable Software*, pages 36–42. IEEE Press, 2015.

[33] Michaela Rindt, Timo Kehrer, and Udo Kelter. Automatic generation of consistency-preserving edit operations for MDE tools. In *Proceedings of the Demonstrations Track of the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014), Valencia, Spain, October 1st and 2nd, 2014.*, volume 1255 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

[34] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE software*, 20(5):42–45, 2003.

[35] Mary Shaw. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer*, 4(1):1–7, 2002.

[36] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[37] Daniel Strüber, Kristopher Born, Kanwal Daud Gill, Raffaela Groner, Timo Kehrer, Manuel Ohrndorf, and Matthias Tichy. Henshin: A usability-focused framework for emf model transformation development. In *International Conference on Graph Transformation*, pages 196–208. Springer, 2017.

[38] Daniel Strüber, Alexandru Burdusel, Stefan John, and Steffen Zschaler. Henshin: A model transformation language and its use for search-based model optimisation in mdeoptimiser. *Modellierung 2018*, 2018.

[39] Birgit Vogel-Heuser, Stefan Feldmann, Jens Folmer, Jan Ladiges, Alexander Fay, Sascha Lity, Matthias Tichy, Matthias Kowal, Ina Schaefer, Christopher Haubeck, et al. Selected challenges of software evolution for automated production systems. In *13th IEEE International Conference on Industrial Informatics (INDIN)*, pages 314–321. IEEE, 2015.