1 Overall Objective

What is needed to produce {software + artifacts} sustainably?

To fully explain this research objective, we will decompose each constituent part in turn. In the above objective, *software* consists of either programs, which run (execute) on a computer, or libraries, which provide services to be used by programs. Both programs and libraries are created using computer code, written in a programming language. Many developers place most, if not all, of their emphasis on the computer code that makes up programs and libraries. However, we consider the outcome of the development process to be more than just software and the associated computer code; we consider the products of the development process to be {software + artifacts}. The *artifacts* include code, but also all the other potential work products, such as documentation, build scripts, test cases, contributor's guides, etc. From this point onward, we will use the term *softifacts* when we are referencing the combination of software and the associated artifacts. As we will explain below, assessing the quality of sustainability is not possible by only considering the computer code; it depends on all of the artifacts, and on the processes and methodologies used during software development. [Can we consider software as both the computer code and the executable (program or library), or should the term software just mean the executable? That is, when we mean code, should we be careful to not instead say software? —SS]

We define sustainable softifacts, as follows: *Sustainable softifacts satisfy, with a reasonable amount of effort, the software requirements for the present, while also being maintainable, reusable and reproducible for the future.* This definition is a specialization of the general definition of sustainable development from Brundtland (1987), as discussed in Section 3. The notion of "reasonable effort" is part of the definition of sustainability, but reasonable cannot be completely defined because what is reasonable is going to depend on the context. Problems that are large, complex and/or critical are going to have a greater "budget" of time and energy. When comparing different alternatives for processes and methodologies, preference would be given to the alternative that costs less effort, assuming that this alternative achieves the same (or a higher) level of sustainability as its competition. Achieving a reasonable effort means that sustainability depends on the productivity the adopted development and maintenance process. The assumptions listed in the next section (Section 3) outline the typical space of problems where it sustainable softifacts are necessary, and what problem constraints suggest the possibility of a reasonable level of effort.

The overall objectives asks what is needed for produce softifacts sustainably. The *what* is found by answering two questions:

1. What softifacts, in addition to the code, are needed?
2. What software engineering principles, processes and methodologies should be employed?

To determine what softifacts are necessary for sustainability, we need to determine what knowledge needs to be captured and documented. The knowledge comes from the problem domain, from computing, from documentation rules and from the development process. We also need to determine how to document the softifacts. That is, what information should appear in which artifacts?, how should it be formatted and structured?

The adopted software engineering principles, processes and methodologies significantly contribute to sustainability. We exclude tools from this list because, although tools are needed to support processes and methodologies, tools change rapidly. Our goal is to be general. If the right principles, processes and methodologies are found, the tools will adapt to match. However, understanding the current state of the practice will involve looking at tools, as shown in the later list of research questions (Section 3.3). For sustainability to be achieved, the productivity of the adopted process must be considered. The process should lead to high quality artifacts with, as mentioned earlier, a reasonable amount of effort.

To be successful, the process, methods and tools, need:

to avoid repetition (knowledge should only be entered once) reduce/remove drudgery provide flexibility for change I think there are more things in this list ...

This leads to our hypothesis: We can use a process that uses knowledge capture and artifact generation to avoid repetition, reduce drudgery and provide flexibility. We have been calling this process MDE, although I'm not sure @carette is happy with that terminology. For simplicity though, I'll use that term in this message.

If MDE satisfies the qualities of a sustainable softifact generating process, then following the bullet points backwards supports the argument that we can produce softifacts sustainably using MDE. This argument is just on the process though; MDE doesn't really dictate what artifacts we want. That is a separate research question.

- overlap plus duplication - required and desired, but still want to be productive. To be productive, want to say once and view as needed. Suggests generation - maybe put this elsewhere?

The objective is not phrased to target a specific software domain, but one of the research assumptions (Section 3) is that we will focus on Research Software (RS) examples and applications. This focus is because creating sustainable softifacts is more feasible when the software domain is well understood, which is the case for RS. Moreover, RS is an important, often neglected, domain of software.

Traditional approach struggles. Try generative. Does generative supply what is needed? Is there an ideal process? - what is needed to produce softifacts sustainably motivates our ideal process (sequence of choices).

Further information on the relevant definitions and context is given in the next section. Roadmap. Research questions for softifacts and knowledge. Research questions for the process.

2 Good Research Questions

The criteria for a good research question are :

- Focused on a single problem or issue
- Researchable using primary and/or secondary sources
- Feasible to answer within the timeframe and practical constraints
- Specific enough to answer thoroughly
- Complex enough to develop the answer over the space of a paper or thesis
- Relevant to your field of study and/or society more broadly

3 Context, Definitions and Assumptions

The overall objective is too ambiguous as given. It requires additional information/interpretation. Therefore this section provides definitions of the relevant terms and a list of the assumptions that define the potential contexts where sustainable softifacts development is reasonable.

3.1 Definitions

Knowledge Facts, definitions, theories, assumptions, etc. [needs a proper definition, with citation(s) —SS]

"Awareness or familiarity gained by experience of a fact or situation." (?) [Definition from Lexico.com (Oxford) dictionary —AD]

Process Organization of the software development activities. [needs a proper definition, with citation(s) —SS]

"A series of actions or steps taken in order to achieve a particular end."(?) [Definition from Lexico.com (Oxford) dictionary —AD]

Principles Principles are "general and abstract statements describing desirable properties of software processes and products." (Ghezzi et al., 2003, p. 41)

Methods "Methods are general guidelines that govern the execution of some activity; they are rigorous, systematic, and disciplined approaches." (Ghezzi et al., 2003, p. 41)

Techniques "Techniques are more technical and mechanical than methods." (Ghezzi et al., 2003, p. 41)

Methodology Combination of methods and techniques. (Ghezzi et al., 2003, p. 41)

Tools "Tools ... are developed to support the application of techniques, method and methodologies." (Ghezzi et al., 2003, p. 41)

Scientific Computing the use of computational tools to analyze or simulate (continuous) mathematical models of real world systems of engineering or scientific importance so that we can better understand and (potentially) predict the system's behaviour. (Smith and Lai, 2005)

Sustainable We start with the general definition:

> "Sustainable development is development that meets the needs of the present without compromising the ability of future generations to meet their own needs." (Brundtland, 1987)

This definition is general; it is not specific to software and it is silent on the specific needs of the present and the future. The definition can be made less abstract by introducing the context of softifacts and thinking separately about the needs of the present and the future. For the needs of the present, the software should meet its requirements specification. [highlight correctness? reliability? —SS] For the needs of the future, given uncertainty, this means supporting change from relevant concerns (political, economic, social, technical, legal and environmental) should be possible with a reasonable level of resources. Supporting the future then means producing softifacts that are maintainable, reusable and reproducible.

Summarizing the above, the proposed definition for sustainability is:

Sustainable softifacts satisfy, for a reasonable amount of energy, the software requirements for the present, while also being maintainable, reusable and reproducible for the future.

This definition has brought in several other qualities. These qualities are defined in the "Quality Definitions of Qualities" document. [Reproducibility above might be changed to replicability, or possibly both terms should be included? —SS] [Should performance be included in the list of qualities? —SS] [Almost every SCS needs to be correct, reliable and usable, but many of them may not need to provide edge performance. How do we choose which qualities to be included here? —AD]

Software Application and/or library.

Application (App) "Software designed to fulfill specific needs of a user." (IEEE, 1991)

"Application software is a program or group of programs designed for end users. These programs are divided into two classes: system software and application software. While system software consists of low-level programs that interact with computers at a basic level, application software resides above system software and includes applications such as database programs, word processors and spreadsheets. Application software may be bundled with system software or published alone."(?) [Definition from Techopedia.com —AD]

"A program or piece of software designed and written to fulfill a particular purpose of the user."(?)[Definition from Lexico.com (Oxford) dictionary —AD]

Library Services that are available to other programs, but not an executable program itself. [needs a proper definition, with citation(s) —SS]

"A software library is a suite of data and programming code that is used to develop software programs and applications. It is designed to assist both the programmer and the programming language compiler in building and executing software."(?)[Definition from Techopedia.com —AD]

"A collection of programs and software packages made generally available, often loaded and stored on disk for immediate use."(?)[Definition from Lexico.com (Oxford) dictionary —AD]

Artifacts Work products generated during the process of creating software. The work products include requirements documentation, design documentation, verification and validation plans, verification and validation reports, contributor's guides, user guides, build scripts, code, test cases, etc.

3.2 Assumptions

Producing sustainable softifacts is not a trivial undertaking. Not every project needs to aim for sustainability. Our position is that the time and energy is justified when one or more of the following assumptions applies.
- number or name assumptions

1. The software project is going to be long lived, where long life means at least 10 years.

2. The softifacts should interest multiple stakeholders (not just the original developer), with different interests and backgrounds.

3. The software is safety relevant, such as software for nuclear safety, medical imaging or computational medicine.

4. [this one is about feasibility - separate this out —SS] The software explicitly, or implicitly, is part of a program family of related software. When the program family assumption means that there are related programs that are less effort to design and build together than to design and build as separate projects. Program family development depends on satisfying three hypotheses (Weiss, 1997):

   - Redevelopment hypothesis – most software development involved in producing an individual family members should be redevelopment because the family members have so much in common.
   - Oracle Hypothesis – the changes that are likely to occur during the soft- ware's [software's —AD] lifetime are predictable.
   - Organizational Hypothesis – designers can organize the software and the development effort so that predicted changes can be made independently.

5. The project recognizes that software artifacts other than the code are relevant.

6. The domain of the software project is well understood. This assumption is related to the oracle hypothesis. [related to feasibility —SS]

- program family comes up to make it economically feasible, not necessary, but important (JC gave example of space probe - not a family, but still makes sense to do it right.) - sustainable - part of reasonable effort Although this study is not specifically restricted to open-source software, for practical reasons (especially the lack of access to the code), commercial software will not be strongly emphasized.

3.3 Current State of the Practice Research Questions

A good starting point for understanding how to develop sustainable SCS softifacts is to look at how SCS is currently developed. By measuring relevant qualities and relating them to development practices, we can form an idea of what is currently working well.

Following the key points of the overall objective, the research questions are based on investigating the following: i) knowledge, ii) principles, processes and methodologies, iii) software qualities and iv) the necessary investment of time and energy. The scope of the research questions are not trying to cover everything, but rather to focus on the most important aspects for SCS.

Define what is meant by meaningful (what criteria need to be satisfied), since this term is used several times in the research questions.

1. What knowledge is currently captured in the artifacts generated by existing open source SCS projects?

2. What principles, processes, methodologies and tools are used by existing open source SCS projects?

3. What are the "pain points" for developers working on SCS projects? What aspects of the existing processes, methodologies and tools do they consider could potentially be improved?

4. For a given software product (softifacts), how can we produce a reasonable measure of the software qualities (correctness, reliability, usability etc.) with a few hours of effort?

5. For a given software product (softifacts), how can we produce a meaningful measure of usability with a few days worth of effort?

6. For a given software product (softifacts), how can we produce a meaningful measure of maintainability (with respect to likely changes) with a few days worth of effort?

7. For a given software product (softifacts), how can we produce a meaningful measure of reproducibility/replicability with a few days worth of effort?

8. With a reasonable effort and time, how can we compare the productivity between different processes, methodologies and tools?

9. Can a correlation be identified between software projects that use best practices for the development process and methodologies and improvement in the qualities of usability, performance and reproducibility/replicability?

3.4 Ideal Process

Some empirical research studies of SCS stop at the previous step. They aim to identify what is currently working well, and then implicitly assume that the identified process/methodologies/tools provide the best guidelines for others to follow. However, identifying the best current approach to developing SCS does not imply that the best possible approach has been found. Our instinct is that there is plenty of room for improvement with SCS development. The following research questions explore this possibility.

1. Ignoring the time and effort required, what knowledge is necessary to capture for sustainable SCS? How can this knowledge best be documented in software artifacts?

2. How can Model Driven Engineering and Code Generation be fit in the development process for SCS to remove as many repetitive, tedious and error prone tasks as possible? We will call this the "ideal process."

3. How well do existing modelling and code generation tools do at implementing the "ideal process"? Candidates for comparison include Drasil and Epsilon.

4. How well does a model driven approach to compared to a traditional approach for usability, productivity and sustainability?

5. What are the attitudes and preferences of typical users towards an implementation of the "ideal process"?

References

G. H. Brundtland. *Our Common Future*. Oxford University Press, 1987. URL https://EconPapers.repec.org/RePEc:oxp:obooks:9780192820808.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

IEEE. Ieee standard glossary of software engineering terminology. Standard, IEEE, 1991.

W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ȧgerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

David M. Weiss. Defining families: The commonality analysis. *Submitted to IEEE Transactions on Software Engineering*, 1997. URL http://www.research.avayalabs.com/user/weiss/Publications.html.