Review of Software Sustainability of global impact models

Paper by Nyenah et al.

Review by Rolf Hut

The authors set out to study how sustainable (the software of) a collection of global impact models is and to provide the readership with hands-on advice on how to improve software sustainability. This work is an important addition to our (meta-)knowledge about the (software behind the) models that we use. I fits the scope of the journal and is well written.

I do, however, want to raise some points that I would invite the authors to respond to. Most of these points are more starters for discussion than concrete suggestions for how to change the paper.

Major (including philosophical) point

On purpose of the work

The work hinges on two thoughts: on the one hand do the authors conduct a study into the state of software sustainability and present their findings. This is a subject fitting of a research paper. On the other hand the authors write an opinion paper on how they advise the community to improve the sustainability of their software. While not made explicit in the paper, it seems (to me) that the authors use the analysis to justify the need for the advice given, for example: “in software development a best practice is to write (good) documentation. From our analysis we conclude that good documentation is lacking for X % of the GIMs studied. We therefore urgently advice the community to write better documentation.”. Although not made explicit in the article, I think this is a good logical structure for an opinion article like this. My main concern is that of the six advice given three (maybe four) are covered by the questions and results of the analysis. The need for the other advice is not justified by the analysis.

- “apply project management” → not covered by survey (and see ‘on agile’ below).
- “consider software architecture” → maybe covered by analyses of lines of code per file, but hard to justify given different software architectures
- “select license & consider version control” → covered by analysis
- “improve code standards / test” → partially covered by analysis
- “make documentation comprehensible” → partially covered by analysis: quality is not checked, only if documentation is available at all.
- “Engage community” → marginally covered by analysis of active number of developers.

For more clarity and overall a stronger point towards the community I would restructure the paper to follow this logic:

1. We know that these advices are good practices in software design. (Make clear if the source of this best practice is literature or the lived experience of the authors themselves.)
2. To pinpoint the need for these advices we conduct an analysis that studies KPIs that signal of these advices are already implemented
3. For 3 out of 6 of these advices we find evidence that these are not being implemented
4. For the other advices, based on our lived experience, we think those are not being implemented
5. Science would be better if they were implemented broadly.

(note: this is an advice on how I would restructure the paper, I am looking forward to the reply of the authors with potential different viewpoints and I certainly do not say that restructuring like suggested is a "must" before the paper can be published).

On purpose of research software

The authors state that impact model "provide crucial information for policymakers, scientists and citizens" (line 38). Further on they argue that "research software that suffers from these shortcomings [….] impede research progress, decrease research efficiency and hinder scientific progress" (line 63). The citizens and policymakers seem to be out of the picture here. The point I want to raise is that different GIMs are made for different audiences. Most GIMs are made by scientists for scientists. The output of the model runs, the conclusions of the papers, can impact policymakers, but the model itself is not intended to be run, or analysed, by them. This contrasts with a number of GIMs that are used by policymakers (and / or citizens) themselves. For example, in my field (hydrology) the PCRGlobWB model (one of the GIMs in this paper) made by Utrecht University is used in academic research. On the other hand, the WFLOW model made by Deltares is both used by Deltares in consultancy work, it is run by operational institutes (governments) as part of their daily operations and it is used in academic publications. Organisations like Deltares (and DHI & SMHI for exampale) have dedicated software design teams. I wonder if the software behind models like this are at a different quality level compared to purely academic software. I would like to ask the authors to reflect on this and what it means for the interpretation of their analysis results.

On representativeness of the selected GIMs

The authors acknowledge the bias introduced by the selection mechanism of GIMs for their analysis. I think that the implications of this bias stretch further than the considerations they authors give in their 'limitations' section. The amount of 'legacy code' differs greatly between different fields of science. Not long ago (I'm getting old…) it was perfectly acceptable to make software available through "please email me". These models are still used, even as GIMs. By using this selection, not only does that exclude some fields of science, but it also selects on

- The type of programming language used at the time
- The type of code quality and style of documentation

Because of this, I'm afraid that the analysis of the authors shows a "best case" situation of the state of software quality in the field, which makes their advice all the more urgent.

On costs and rewards

The authors conclude that the quality of research software would improve if their advice is followed and they also conclude that writing and maintaining complex research software is expensive (line 386). The obvious questions that the authors do not address are: why is this as it is and who should pay for the change? To answer the first question, I would argue that scientists do not have an incentive to make their software better. A Good paper with bad software gets you just as far and writing bad code is much faster than nicely documenting and structuring your work (I have been guilty of this myself). Just asking scientist to "do better" will not change if those that do better are not rewarded for these efforts (or, more negatively, those providing bad code punished with negative career outcomes). I invite the authors to reflect on this and provide their vision on how to arrive on a situation where researchers feel they have the time, the budget and the incentives to actually spend time on writing better research software.

On agile

One of the advices that the authors give and that I want to push back on is to "apply project management practices" and in particular "Agile" when building research software. The 'agile' framework for developing software originated in the start-up culture of silicon valley and it is laced with assumptions that the environment that one works in is that of an anglo-saxon, shareholder value driven, software start-up. The main idea of Agile as I distil it is for a team at any point in time to decide on the next action that optimizes the value of the software product (sic) most. The underlying assumption are:

- It is unclear what the final product should be, we develop as we go, requirements can change any moment
- We have an existing product and an existing user base (customers) that we can continuously test our improvements with
- The crucial limit is the amount of development time (cost).

Especially the first point is plainly not true in many scientific projects such as big EU projects where it is decided upfront what should be delivered and requirements are fixed, yet budgets are not (always). (This is partly why many software projects in government and big institutes always go over budget).

I hope the authors recognize that I strongly push back against advocating for agile and would be much happier with something akin to: "choose a project management practice that honors the boundary conditions of the institutional environment and culture you are part of". I'm looking forward to reading the authors view on this.

Minor points

On reproducibility through platforms

With the risk of sounding like a reviewer that is mainly pushing their own work: reproducibility does not, per se, have to be fixed by the model developer. There are a few 'model platforms' that have emerged in the last years that give people access to either models itself, or the

output of models, in a FAIR way. This is also a way in which the community as a whole can arrange for better (reproducible) software. Projects that spring to mind from my own field include ESMValTool, eWaterCycle (the one I work on), PAVICS-Hydro, Deltares FEWS. Do the authors agree that this is a way as a community to make research software more reproduceable?

On researchers as software developers

On line 54 the authors claim that "most of these researchers are self-taught software developers with little knowledge of software requirements.". The authors do not back up this claim by referencing literature that has studied this. While I do agree that this might have been true a few decades ago, I would argue that currently at least all scientists get some sort of programming classes in their education and those doing a MSc or similar in "our fields" get quite extensive programming classes. This leads to two questions to the authors:

- do they stand by their claim?
- More related to the conclusion and how to implement the advices that the authors give: what do they think the role of education in general and graduate level eduation in particular should be?

On the software cost estimation

While I recognize the need to somehow quantify the amount of work (money) invested in I strongly think that the number of decimals in the coefficients presented hint at more certainty in the relation than the underlying data that this is fitted to justifies. I would like to ask the authors to do a bit of a sensitivity analyses on these coefficients to see how much their conclusions change. Furthermore, I think that the 18 large NASA projects include those kind of projects where an error in a line of code could mean that the lander misses the moon and astronauts die, where-as an error in a GIM usually means that a future projection is slightly off. (I'm being mean, but I do think that NASA spends more time on code quality control than most post-docs working on GIMs). I'd like to ask the authors to reflect on this a little bit in their discussion.

Really minor points

- Line 44: are non-physical impact models also possible (economical models? Demographic models? Statistical models?)
- Line 81: what about the cost of re-implementing / making reproducible versus the cost of maintaining old code?
- Line 102: I find it somewhat funny that the list of isimip models is itself not a FAIR datasource (no further action required)
- Line 165: could the absence of active developers not also imply very mature software that is just perfect?
- Line 171: singularity is now called apptainer
- Line 230: the references to sloc look weird: is this the preferred format?
- Table 2: I suggest to use '+' instead of 'x' to indicate availability.

- Figure 2: I suggest to add “no license” and “not OSI license” as columns to this figure.
- Figure 5 and figure 6: is it somehow possible to indicate which of these models are in which programming language? For example sort along the x-axis by programing language such that all the python models are next to each other?