Long-Term Productivity Based on Science, not Preference

Spencer Smith
Computing and Software
McMaster University
Canada
smiths@mcmaster.ca

Jacques Carette
Computing and Software
McMaster University
Canada
carette@mcmaster.ca

Our goal is to identify inhibitors and catalysts for productive long-term scientific software development. The inhibitors and catalysts could take the form of processes, tools, techniques, and software artifacts (such as requirements, specifications, user manual, unit tests, system tests, usability tests, build scripts, READMEs, license documents, process documents, and code). The effort (time) invested in catalysts will pay off in the long-term, while inhibitors will have minimal (possibly negative) benefits relative to their cost.

If developers are surveyed on inhibitors and catalysts, their answers will be as varied as the education and experiential backgrounds of the respondents. Their responses will be well-meaning, but they will undoubtedly come with problems and biases. For instance, developers may be guilty of the *sunk cost fallacy*, promoting a technology that they have invested considerable hours in learning, even if the current costs outweigh the benefits. As another example, developers are known to typically not produce proper requirements specifications [1]. If they don't recommend requirements documentation, that isn't proof that this kind of documentation is an inhibitor. Another perceived inhibitor is meetings [**?**], even though some meetings are very useful in the long-term. The trick is to know which meetings are useful, and which are not. As these examples illustrate, we need to take developer and personal preference out of the development process and instead pick the artifacts/processes/tools that have a long-term impact. We need to use a scientific approach based on unambiguous definitions, empirical evidence, hypothesis testing and rigorous processes.

1 BUILDING BLOCKS

A scientific approach requires a solid foundation. A scientific basis built on unambiguous definitions of software qualities, including definitions for the qualities of reliability, sustainability, reproducibility and productivity (along the lines of our quality definitions paper).

- Give the qualities of the definitions we are looking for - okay to not be measurable.

It is still too early to give a directly measurable definitions of qualities. We thus settle on a definition that can at least allow us to reason about each quality. Scientific computing (the heart of much research software) is our model: its definition of (forward) error requires knowing the, usually unknown, true answer.

- Focus on definition of productivity, using our long-term productivity for long-term impact paper [2].

As a starting point, here is our conceptual formula for productivity:

$$I = \int_0^T H(t)\ dt$$

$$O = \int_0^T \sum_{c \in C} S_c(t) K_c(t)\ dt$$

$$P = O/I$$

where P is productivity, I is the inputs, O is the outputs, 0 is the time the project started, T is the time *in the future* where we want to take stock, H is the total number of hours available by all personnel, C represents different classes of users (external as well as internal), S is satisfaction and K is *practical knowledge*. Thus productivity is measured in "satisfying reusable knowledge per hour."

2 MEASURING PRODUCTIVITY

Interpretation of equation from previous section.

Measuring the impact of interventions on productivity (somewhat related to SOP work and Olu's work, but not something we've explored too much to date)

return on investment = (net program benefits / program costs) x 100

We adapt the standard definition of productivity [5], where inputs are labour, but adjust the outputs to be knowledge and user satisfaction, where user satisfaction acts as a proxy for effective quality. This explicit emphasis on all knowledge produced, rather than just the operationalizable knowledge (aka code) implies that human-reusable knowledge, i.e. documentation, is crucial. This is why the long-lived context is important.

3 CURRENT STATE OF THE PRACTICE

Understanding the current state of the practice, including finding a methodology to assess the state of practice (SOP work)

Understanding the gap between what is recommended and what is practiced (Olu's work)

4 WHAT ARTIFACTS?

Determining what documentation should be produced based on what is necessary for an assurance case argument for reliability (my work on assurance cases)

5 WHAT PROCESS?

Removing the drudgery of document generation and maintenance via generation (Drasil)

6 CONCLUDING REMARKS

REFERENCES

[1] Dustin Heaton and Jeffrey C. Carver. 2015. Claims About the Use of Software Engineering Practices in Science. *Inf. Softw. Technol.* 67, C (Nov. 2015), 207–219. https://doi.org/10.1016/j.infsof.2015.07.011

[2] Spencer Smith and Jacques Carette. 2020. Long-term Productivity for Long-term Impact, arXiv report. https://arxiv.org/abs/2009.14015. arXiv:2009.14015 [cs.SE]