Essay: DevOps practices for sustainable software development

Maël Madon madon@kth.se

April 29, 2020

Abstract

Modern societies are relying more and more on software. At the same time, the impact of new technologies on the environment is growing. We discuss in this essay the opportunities for DevOps practices to help the development of more sustainable software. We identify that DevOps approach and tools for continuous integration, deployment, testing and monitoring have a potentiel to reduce the footprint of the development and use phase of softwares. However by being essentially technical, DevOps fails to cause more meaningful and necessary changes in the industry.

1 Introduction and background

The world we are living in is facing a dangerous ecological crisis. The 2018 report from the Intergovernmental Panel on Climate Change (IPCC) [1] estimates that the average global temperature in 2017 was 1°C above the pre-industrial level. A global warming of 1.5°C is very likely to increase sea level rise, extreme hot events, heavy precipitations, loss of biodiversity, etc. In order to limit global warming below 1.5°C, global net anthropogenic CO_2 emissions should decline by about 45% from 2010 levels by 2030, reaching net zero emissions by 2050. It represents a major shift in the economy and requires a tremendous effort from every. The challenge is to become sustainable i.e. to act such that "the use of renewable resources [proceeds] at a rate that is less than or equal to the rate of natural replenishment" [2]. The Information and Communication Technology (ICT) sector is no exception.

ICT ecological responsibility According to this study [3], the contribution of the ICT sector to the global greenhouse gases emissions has almost tripled from 1.06–1.6 % in 2007 to 3.06–3.6 % in 2020. Hilty et Aebischer classify in [4] ICT impacts on the environment into three categories: (i) direct (negative) effects through infrastructure and energy consumption, (ii) second-order "enabling" effects that can be positive (optimization, dematerialization) or negative (obsolescence, induction of the consumption of a new resource); (iii) third-order "systemic" effects though the evolution of behaviors, consumption patterns and organization. The last effects are longer-term impacts and the most difficult to predict. However it is essential to keep them in mind to have an accurate picture when taking decisions.

Sustainable software engineering Taking decisions is exactly what happen everyday in

software development companies. These decisions can have direct or indirect effect on the environment. The signatories of the Karlskrona manifesto for sustainability design [5] deplore that long-term sustainability goals in software engineering are most of the time relegated to the second place behind short-term success criteria such as user satisfaction or reduced costs. It is even more regrettable giving the place that software systems have taken today in shaping the world. The manifesto argues for a more holistic approach in the software community in order to shift sustainability from being just a *system quality* to a primary *concern*. *Sustainable software engineering* is defined in [6] as "the art of defining and developing software products in a way, so that the negative and positive impacts on sustainable development that result and/or are expected to result from the software product over its whole life cycle are continuously assessed, documented, and used for a further optimization of the software product".

Agile practices for green software development Considering this definition, developing sustainable software is a lot about practices and methods. For example the potential of the agile methods [7], more and more popular in software engineering, have been studied in the literature. The purpose of agile software development is to produce fast remarkable software, at low cost and with high user satisfaction. Rashid et Khan conducted in [8] a systematic literature review and an industry survey and identified six critical success factors with their corresponding agile practices that can help the development of sustainable software.

In a similar way as in this study, we can wonder which *DevOps* practices are likely to help in this regard. We will discuss in this essay the following question: what are the possible interactions and synergies between DevOps practices and the development of sustainable software? We will start in Section 2.1 by identifying a lack of research in this topic. Section 2.2 will give a personal reflection about potential interactions. Finally we will explain in Section 2.3 why DevOps alone is not a sufficient approach to reach the necessary level of sustainability.

2 Discussion

2.1 Lack of research in the topic

Our first idea for this work was to adapt the methodology of systematic literature review (SLR) as presented in the guidelines [9] and conduct a mock SLR on our research question. As it was not the purpose of an essay, this idea was abandoned for a more personal reflection instead. However, running a search string in a scientific search engine appeared interesting to highlight the lack of research in the topic.

Rashid et Khan in their SLR on agile and sustainable software built the following search string [8]:

("Green software" OR "Greener software" OR "Sustainable software" OR "Green software engineering") AND ("Agile" OR "Agile methods" OR "green agile") AND ("Practices" OR "Solutions")

Which can actually be simplified given the rules of search engines in

("Green software" OR "Greener software" OR "Sustainable software") AND "agile" AND ("Practices" OR "Solutions")

At the time of the study (2015), 374 publications were found in Google Scholar. The same search string today in Google Scholar gives 848 results (04/27/2020). We replaced the keyword "agile" by "devops" and ran the search again in Google Scholar. This time, only 97 results were obtained, all of them except a few having less than 5 citations. By quickly screening the title and abstracts, we found that no article was in direct relation with our research question.

Of course, the literature review should not stop here and the keyword "devops" is likely to be to restrictive to find relevant publications. Splitting it into subtopics like "deployment", "automatic testing", "continuous integration" would be needed, as well as extending "sustainable software" to "energy saving" or "environmental impacts". Yet it shows that to the best of our knowledge nobody has tackled the links between devops practices and sustainable software as a whole.

2.2 DevOps practices encouraging the development of sustainable software

Where does the potential lie for a DevOps practitioner then? We give in this section personal thoughts providing answers to this question. With the help of [10] and the "topic" label in the GitHub issues of the course repository[1], we picked the five following DevOps practices and discuss thereafter their interactions with sustainable software: continuous integration, deployment ans infrastructure, testing and monitoring. These practices won't be defined in details as we assume they are already known by the reader.

[1] github.com/KTH/devops-course/labels/topic

2.2.1 Continuous integration

One fundamental practice of modern software development bridging the gap between developers and operations is continuous integration (CI). Having small commits, automated building and testing make the program easier to debug and understand. Software developers avoid spending days solving "merging hell". CI tools such as Jenkins or Travis can be configured to enforce discipline in the developers' contributions by automatically blocking pull requests if certain criteria are not met. Overall, continuous integration improves the quality of the software. It accelerates the development process while improving collaboration and reviewing. This practice is also advocated by agile practitioners and if it has not been made for environmental purposes, it contributes to building sustainable software through an "efficient utilization of time and computing resources", which has been identified in [8] as a critical success factor.

However, more than a simple *consequence* of the efficiency it brings, we believe that CI can play a more active role in building sustainable software. By reflecting on that, we try to follow the Karlskrona manifesto [5] and switch sustainability from being a "good to have" quality to a primary concern. For example, usual metrics that are computed in CI pipelines are test coverage or code complexity. They are displayed as immediate feedback to developers. They can also be used to enforce a certain discipline by blocking for instance the pull request if one method is not documented, too big or untested. In a similar way, sustainability metrics could be computed and used in the CI pipeline. If could be a benchmark checking that a function is not using too much resource (CPU, memory, network, runtime...). It could be a tool for estimating the

energy consumption of the software as the one developed in [11]. More generally, researchers proposed in [12] a quality model for sustainable software. All the criteria from this model that can be automated deserve to be reported and checked during the development phase.

2.2.2 Deployment and infrastructure

Building the infrastructure and the deployment strategy for a software may be where the DevOps engineer has the most impact on the environment. For example, Verdecchia et al. have studied in [13] the energy impact of different deployment strategies for a software. If the paper can not conclude in an absolute preferable strategy with respect to energy consumption, it highlights that the result can differ significantly depending on the alternative chosen. In fact: deciding to dedicate one or two machines for a task or allocate a specific amount of memory for a service will affect the consumption of underlying hardware.

In the DevOps world, the tendency is to deploy in the cloud and let orchestration tools and load balancers manage the traffic and the allocation of resources. It has a rather positive effect on energy consumption as it relies on the energy efficiency and source of supply of the cloud infrastructure and the quality of its scheduling where lot of optimization have been done. Software vendors can also choose between a wider and wider variety of "green cloud" providers.

One could argue that the cloud being "invisible" compared to on-premise equipment would make the developer care less about oversizing or switching off services that are not used. But in practice, the "pay-as-you-go" pricing by cloud providers offers a financial incentive to save computing resources that is in line with environmental objectives. DevOps engineers should then include in their decisions this environmental cost. A good example is comparing two usual deployment policies: blue/green deployment and canary testing. The first one consists of keeping two complete environment ready: a "blue" environment, in production, and a "green" environment, only accessible for development and testing. Updates and rollbacks can be done easily by routing the traffic to one or the other. This solution has a higher infrastructure cost than canary testing in which we deploy the new update only by routing a small proportion of the traffic (typically 5–10%) to the new release and eventually all of it if everything works well.

2.2.3 Testing

We already touched upon testing in the previous subsections. As discussed for continuous integration, testing also improves the overall quality of the software. Automated testing and in particular systematic unit testing are practices associated with DevOps that speed up the process of finding bugs, make developers responsible for the functions they write and contribute to a clearer code. This leads to efficient development of remarkable software, which is part of the definition of sustainable software engineering. It also contributes to the release of robust software, that can last long and resist to security threats. It relates to the question of longevity we have not yet addressed but which is closely linked with sustainability.

Similarly to deployment strategies, different testing strategies need different infrastructures that can be more or less energy intensive. Testing as well as security is often seen as a trade-off between having a robust and reliable software and spending time and resources in integra-

tion testing, A/B testing, fuzz testing, etc.. The question should be asked if heavy black-box testing constantly requiring thousands of machines is really worth it to reach one percent more coverage or to find a few bugs.

2.2.4 Monitoring

Finally, we are going to discuss in this section about a crucial DevOps activity: monitoring. If we come back to the definition of sustainable software engineering stated in the introduction, we realize that monitoring has an important role to play. If it does not necessarily report negative and positive environmental impacts of the software product over its whole life cycle, it is meant to give feedback on the use phase of the product. We believe that a big margin of progress lies in this domain to include green metrics and analytics in popular monitoring tools.

Nevertheless, monitoring is worthless if the data extracted stays unused or does not lead to decisions. This invites us to take a step back and wonder what is the room for such a technical work as DevOps to induce meaningful changes. This is the topic of next section.

2.3 DevOps and sustainability: not the right angle?

To conclude this discussion, one can notice that DevOps practices being essentially technical, they fail to bring something more than technocentric solutions to mitigate ICT impact on the environment. It is not only insufficient but also dangerous as it makes us believe we are on the right track and doing our best. The signatories of the Karlskrona manifesto warn their community against this threat: "There is a tendency to think that taking small steps towards sustainability is sufficient, appropriate, and acceptable. Whereas incremental approaches can end up reinforcing existing behaviours and lure us into a false sense of security. However, current society is so far from sustainability that deeper transformative changes are needed." [5]. In other words, it seems that DevOps only has the power to target first-order environmental effects as categorized in [4] and explained in introduction. That may explain the lack of literature dealing with the links between this set of practices and sustainable software as compared to agile practices. Indeed, if agile methods are an older concept they are also a more holistic approach of the development process including human management and organization of the project.

DevOps practices can technically help to develop more light weight applications, make them portable, more easy to maintain. It also encourages efficient collaboration though the sharing of configuration files, tutorials, videos, examples. This appears as a necessary stone for resilient, robust and maintainable software. But it stays only a set of tools with no control over the very purpose of the software being developed. It does not question its usefulness. Moreover, it has been recognized that advances in energy efficiency of technologies almost always lead to a growth in these technologies that tend to compensate the positive effect. This effect is called "rebound effect" or "Jevon's paradox" and ICT is no exception to this rule [14]: the current tendency is that people own and use more and more devices in their everyday life. If these issues are obviously out of the scope of this essay, we think it is important to mention them and keep them in mind.

3 Conclusion

This essay discussed possible interaction with DevOps practices and the development of sustainable software. Our conclusion is that DevOps has powerful tools to help in the development, testing, monitoring and deployment of sustainable software. We identified relevant practices and their links to energy efficiency that deserve in our opinion to be accurately quantified in future research. We also tried to draw the boundary between what DevOps can be expected to achieve and what is out of its scope. In particular, we strongly agree with the Karlskrona manifesto for sustainable design [5] to recognize that it is necessary to question ourselves on the impact of the technologies we develop. This is not only a matter of technical solutions but rather a reflection about the path in which we want to lead society.

Acknowledgments

This essay has been written as part of a course assignment for KTH *DD2482 Automated Software Testing and DevOps* course. The essay proposition can be viewed in this pull request: github.com/KTH/devops-course/pull/395. I thank the TAs for their advice in the github discussion as well as during lab sessions.

References

[1] V. Masson-Delmotte et al. *Global Warming of 1.5C*. IPCC, 2018.

[2] Richard Heinberg and Daniel Lerch. "What Is Sustainability?" In: *The Post Carbon Reader: Managing the 21st Century's Sustainability Crises*. Healdsburg, CA: Watershed Media. 2010.

[3] Lotfi Belkhir and Ahmed Elmeligi. "Assessing ICT Global Emissions Footprint: Trends to 2040 & Recommendations". In: *Journal of Cleaner Production* 177 (Mar. 2018), pp. 448–463. ISSN: 09596526.

[4] Lorenz M. Hilty and Bernard Aebischer. "Ict for Sustainability: An Emerging Research Field". In: *ICT Innovations for Sustainability*. Springer, 2015, pp. 3–36.

[5] Christoph Becker et al. "Sustainability Design and Software: The Karlskrona Manifesto". In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE). Florence, Italy: IEEE, May 2015, pp. 467–476. ISBN: 978-1-4799-1934-5.

[6] Stefan Naumann et al. "The GREENSOFT Model: A Reference Model for Green and Sustainable Software and Its Engineering". In: *Sustainable Computing: Informatics and Systems* 1.4 (Dec. 1, 2011), pp. 294–304. ISSN: 2210-5379.

[7] Kent Beck et al. "Manifesto for Agile Software Development". In: (2001).

[8] Nasir Rashid and Siffat Ullah Khan. "Agile Practices for Global Software Development Vendors in the Development of Green and Sustainable Software". In: *Journal of Software: Evolution and Process* 30.10 (2018). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.196 e1964. ISSN: 2047-7481.

[9] Barbara Kitchenham and Stuart Charters. "Guidelines for Performing Systematic Literature Reviews in Software Engineering". In: (2007).

[10] Ramtin Jabbari et al. “What Is DevOps? A Systematic Mapping Study on Definitions and Practices”. In: *Proceedings of the Scientific Workshop Proceedings of XP2016*. 2016, pp. 1–11.

[11] Nadine Amsel and Bill Tomlinson. “Green Tracker: A Tool for Estimating the Energy Consumption of Software”. In: *Proceedings of the 28th of the International Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA ’10*. The 28th of the International Conference Extended Abstracts. Atlanta, Georgia, USA: ACM Press, 2010, p. 3337. ISBN: 978-1-60558-930-5.

[12] Stefan Naumann et al. “Sustainable Software Engineering: Process and Quality Models, Life Cycle, and Social Aspects”. In: *ICT Innovations for Sustainability*. Springer, 2015, pp. 191–205.

[13] Roberto Verdecchia et al. “Estimating Energy Impact of Software Releases and Deployment Strategies: The KPMG Case Study”. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). Nov. 2017, pp. 257–266.

[14] Cédric Gossart. “Rebound Effects and ICT: A Review of the Literature”. In: *ICT Innovations for Sustainability*. Springer, 2015, pp. 435–448.