Embracing Deep Variability For Reproducibility and Replicability

Mathieu Acher, Benoît Combemale, Georges Aaron Randrianaina, Jean-Marc Jézéquel

Embracing Deep Variability For Reproducibility & Replicability

Mathieu Acher
Univ Rennes, Inria, CNRS, IRISA, IUF
Rennes, France

Benoit Combemale
Univ Rennes, Inria, CNRS, IRISA
Rennes, France

Georges Aaron Randrianaina
Univ Rennes, Inria, CNRS, IRISA
Rennes, France

Jean-Marc Jézéquel
Univ Rennes, Inria, CNRS, IRISA, IUF
Rennes, France

ABSTRACT

Reproducibility (*a.k.a.*, determinism in some cases) constitutes a fundamental aspect in various fields of computer science, such as floating-point computations in numerical analysis and simulation, concurrency models in parallelism, reproducible builds for third parties integration and packaging, and containerization for execution environments. These concepts, while pervasive across diverse concerns, often exhibit intricate inter-dependencies, making it challenging to achieve a comprehensive understanding. In this short and vision paper we delve into the application of software engineering techniques, specifically variability management, to systematically identify and explicit points of variability that may give rise to reproducibility issues (*e.g.*, language, libraries, compiler, virtual machine, OS, environment variables, *etc.*). The primary objectives are: i) gaining insights into the variability layers and their possible interactions, ii) capturing and documenting configurations for the sake of reproducibility, and iii) exploring diverse configurations to replicate, and hence validate and ensure the robustness of results. By adopting these methodologies, we aim to address the complexities associated with reproducibility and replicability in modern software systems and environments, facilitating a more comprehensive and nuanced perspective on these critical aspects.

1 INTRODUCTION

Many scientific domains need to process large amount of data with more and more complex computations. For instance, studies about climate modelling and change involve the design of mathematical model, the mining and analysis of data, the executions of large simulations, *etc.* [16, 24, 29]. These computational tasks rely on various kinds of software, from a set of scripts to automate the deployment to a comprehensive system containing several features that help researchers exploring various hypotheses. It is not an overstatement to say that computational science depends on software and its engineering [2, 34, 54].

One of the main promise of software is that a result obtained by an experiment (*e.g.*, a simulation) can be achieved again with a high degree of agreement. But despite the availability of data and code, several studies report that the same data analyzed with different software can lead to different results [6, 9, 15, 19, 22, 31, 41, 42, 53]. For instance, applications of different analysis pipelines, alterations in software versions, and even changes in operating system have been shown to cause variation in the results of neuroimaging studies [18]. Massonnet *et al.*'s [41] results and experience suggest that earth system models are not replicable under changes in the high-performance computing environment. Similar observations have been made in the machine learning or in the software engineering community [22]. As a result, software can threaten the scientific knowledge and recommendations built on top of these computations and studies.

In this paper we propose to characterize both intended and unintended variability of any software-intensive system in order to support reproducibility and replicability, and eventually estimate its robustness, uncertainty profile, and explore different hypotheses.

2 DEEP SOFTWARE VARIABILITY

Uncertainty in informatics comes from many different origins [17, 39], either ontological (*i.e.*, inherent unpredictability, *e.g.*, aleatory) or epistemic (*i.e.*, due to insufficient knowledge).

Ontological causes include noise in the input data of a program, its memory layout, network delays, the internal state of the processor, the ambient temperature and even the age of the processor[1].

Epistemic causes include misunderstanding of the user's needs, variable behavior of conceptually similar resolution methods, choice of threshold parameters, unexpected behavior of APIs, variable behavior among functionally similar libraries, or subtle differences in the semantics of programming languages (*e.g.*, $-3\%2$ evaluates to -1 in Java but to 1 in Python), or even inside the same programming language (for instance $x/0$ is an undefined behavior in C).

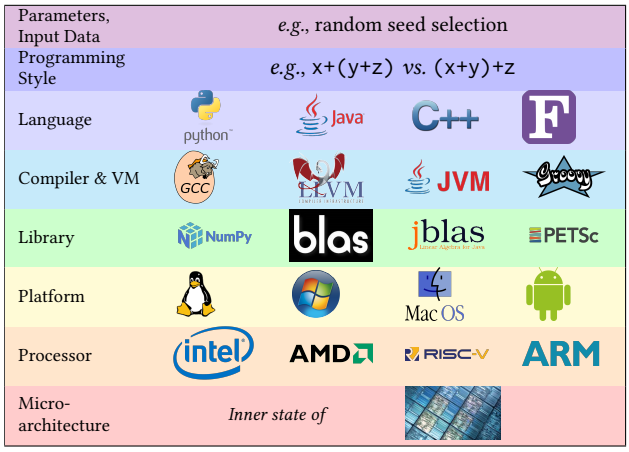

Figure 1: Deep Variability

A further dimension of the problem lies into the fact that these causes of uncertainty might not always be orthogonal. We have

[1] A given processor micro-architecture evolves over time due to failing subsystems that are compensated.

shown in [37] that several layers can interact, *e.g.*, between compile-time and run-time options. In [36] we have coined the term *Deep Variability* to refer to the **interaction of all external layers modifying the behavior (including both functional and non-functional properties) of a software**, as illustrated in Figure 1. Deep software variability challenges practitioners and researchers: the combinatorial explosion of the epistemic and ontological variability complicates the understanding, and thus the design, the configuration, the maintenance, the debug, and the test of software systems.

An illustration. Let's take a simple example to illustrate the problem, concentrating on floating-point numbers even if uncertainty problems go well beyond that. The Python program of Listing 1 first asks for a seed for its pseudo random number generator, then generates 3 pseudo-random numbers x, y, z and checks whether their addition is associative, *i.e.*, $x + (y + z) = (x + y) + z$. When run many times with different seeds, it turns out that this fundamental property of the addition **only holds about 82.8% of the time**.

Code Listing 1: Proportion of pseudo-random numbers for which associativity of addition holds

```
from random import random, seed
def associativity_test() -> bool:
    x = random(); y = random(); z = random()
    return x + (y + z) == (x + y) + z

def proportion(number: int) -> int:
    ok = 0
    for i in range(number):
        ok += associativity_test()
    return ok * 100 // number

seed(int(input('Seed: ')))
print(str(proportion(1000)) + "%")
```

Any programmer unaware of this kind of floating point issues (*a.k.a.*, floating point numbers are not real) is at risk of making mistakes: this is an example of **Epistemic Variability**. One interesting point is that the proportion of cases where the associativity property holds differs greatly depending on the input seed, with extreme cases of 76.2% with a seed=10215250 and 89.2% with a seed=85252568: this is an example of **Input Data Variability** (also related to *stability* in computational learning theory [7]), because here we would have expected that the program would behave the same whatever the input data.

Yet already severe, the problem does not stop here. If we translate this program to Java using `Math.random()` as its random generator, we get that associativity holds 83.1% of the time, which is quite but not exactly similar to Python: this is an example of **Language Variability**. But if we use `java.util.Random.nextFloat()` we get 100%! Two variants of the same functionality ends up behaving differently: this is an example of **Library Variability**.

If we again translate this program to C, compile it with GCC and run it on Linux, we get similar issues depending on which random generator is used (74.6% vs. 100%), but a new dimension appears when we compile the same C program for Windows, now obtaining 99.8% and 100%. This effect is now due to **Platform Variability**. Run on an 32-bits older processor, it yields still different results, which is a manifestation of **Processor Variability**. Experiments on different variability settings (configurations) can be found on our Github repository [1].

Note that there is nothing particularly mysterious here: all these different behaviors can be perfectly understood and rationalized with a good enough understanding of double & floating point arithmetic and an equally good grasp on pseudo-random number generation issues for each of these libraries and platforms. But this might be a little bit too much to ask for say, a geologist or a climate expert or even the average programmer.

3 EVIDENCE OF DEEP VARIABILITY

In this section, we review the literature and provide evidence that variability affects many layers (as presented in Section 2) in many settings and application domains. This deep variability is a threat to reproducibility, but is also a mean to explore different settings and verify the generalizability of knowledge.

Climate model. In the field of climatology, different climate models (*e.g.*, Earth system models *a.k.a.*, ESMs) are being developed based on data and simulations. Owing to the complexity and their importance, some papers explore the reproducibility and replicability of the underlying models. A paper from 2007 [29] varies on purpose both the parameters of the climate models, the hardware, and the underlying software. The authors reported that the hardware has little effects, but the parameters were more problematic with certain values that could potentially question the models developed so far. It is an early example of the use of deep variability to test the generality of a climate model. More recently, Massonnet *et al.* [41] suggest that "the default assumption should be that ESMs are not replicable under changes in the HPC environment, until proven otherwise." Specifically, they report that various variability factors may hamper reproducibility: the way the software is compiled with different compiler flags (*e.g.*, `O2 -g -traceback -fp model strict -xHost`) having an influence on the accuracy of the floating point computations, some specific configuration options for the execution or simply the hardware used as part of a high-performance computing (HPC) since the study requires a lot of computations. The conclusion of their inquiry is that there are scientific studies that are simply not reproducible due to deep variability. That is, you get so many different results because of the sources of variation, that you can only have limited confidence, and therefore the experience is not replicable. As a follow-up, Döscher *et al.* [15] introduced a protocol to explore the variability space. They recommend fixing some variation points to avoid bugs in models that can cause significant changes in the simulated climates. They further warn about hardware and software variability that can potentially affect the ESM climate – these changes may appear unimportant, like the reordering of the call to physical routines, but could profoundly affect the model results. For climate model users, a better understanding of the variability that guarantees the replicability of ESMs is a necessary step to bring more trust.

Machine learning. Wang *et al.* [53] showed that in the field of machine learning, and on Kaggle competitions, altering the versions of Keras and Tensorflow used as part of pipeline can give absolutely different results to the users. There are quite worrying results. Considering the Area Under the Curve (AUC) that goes from 0 to 1, the score can well drop to 0.55, while it is rather at 0.99 on other

versions. That is to say there is a source of variation – here on the versions of libraries – that significantly impact the reliability of your machine learning system. If by bad luck, as an experimenter, as a researcher, as an engineer, you had selected this version, maybe you would have been extremely disappointed with the results, or you would have claimed that Keras and TensorFlow were not suitable for your problem, while in fact it is rather an accidental variability that caused this drop in performance. There are also combinations of versions that lead to crashes and not working systems. Overall, the deep variability here about the version and the configuration of the machine learning pipeline is a threat about effectiveness of some design choices and can question the generality of the proposal and results. Joelle Pineau, in her ICSE 2019 keynote [47], called the community to build reproducible, reusable, and robust machine learning software. Henderson *et al.* [22] precisely listed different questions and variability factors that may affect the generality of results in the field of reinforcement learning. As an experimenter and scientist, here is a list of questions: What is the magnitude of the effect *hyperparameter settings* can have on baseline performance? How does the choice of network architecture for the policy and value function approximation affect performance? How can the *reward scale* affect results? Can *random seeds* drastically alter performance? How do the *environment properties* affect variability in reported algorithm performance? Are commonly used baseline, *implemented in different languages and libraries*, comparable? All these factors are part of the deep variability and resemble to the issues of Section 2 in our simplified example.

Neuroimaging. Glatard *et al.* [18] show the lack of reproducibility of neuroimaging analyses across operating systems, in particular due to floating point arithmetic issues similar to those of Listing 1. Early studies reported on similar issues related to different versions, hardware, and operating systems [9, 19, 31]. The goal of brain imaging is to provide in-vivo measures of the human brain to better understand its functions, structure and connections. Neuroimaging studies are characterized by a very large analysis space. To build their analyses, practitioners must choose between different software, software versions, algorithms, parameters, *etc.* For many years, those choices have been considered as “implementation details" but evidence is growing that they can lead to different and sometimes contradictory results. For instance, the same dataset of functional Magnetic Resonance Imaging (fMRI) results was independently analyzed by 70 teams, testing 9 ex-ante hypotheses [49]. Significant variations appeared in reported results, with substantial effects on scientific conclusions, thus jeopardizing the confidence one could have in these studies. That might ultimately lead to compromising people health, which is a growing concern in this domain [30].

Bluff-body aerodynamics. Mesnard and Barba reported that "completing a full replication study of our previously published findings on bluff-body aerodynamics was harder than we thought. Despite the fact that we have good reproducible-research practices, sharing our code and data openly." [42] The authors reported on specific reasons, that we found are related to variability layers: (1) meshing and bounding conditions (as expressed in some software parameters) can ruin the computational result; (2) all linear algebra libraries are not equal: despite their promise of computing the same thing, implementation details matter and can lead to discrepancies in results; (3) different versions of the code, or external libraries or even compilers may challenge reproducibility. The insights echo with deep variability exposed in Section 2 or reported in other sections and prior works.

Performance modeling of software. Varying software is a powerful means to achieve optimal functional and performance goals. An observation is that only considering the software layer might be naive to tune the performance of the system or test that the functionality behaves correctly [26–28, 32, 38, 46, 51]. In fact, many layers, themselves subject to variability, can alter performances of software configurations. For instance, configuration options may have very different effects on execution time or energy consumption when used with different input data, depending on the way it has been compiled and the hardware on which it is executed.

For instance, run-time options (*e.g.*, command line parameters) interact with compile-time options (*e.g.*, using `./configure`) with different effects of non-functional properties of a software [37]. There are various consequences w.r.t. tuning, default configurations, understanding, and testing of software. The best run-time configuration may not be optimal depending on the way the software has been compiled; the configuration knowledge about the run-time options depends on the compile-time options; a performance prediction model may not generalize and be pointless due to a different compilation. In Lesoil *et al.* [35], a large study over 8 configurable systems quantifies the existing interactions between input data and configurations of software systems. The results exhibit that (1) inputs fed to software systems interact with their configuration options in non-monotonous ways, significantly impacting their performance properties; (2) tuning a software system for its input data makes it possible to multiply its performance by up to ten (3) input variability can jeopardize the relevance of performance predictive models for a field deployment. There are also cases in which both compile-time, runtime, and input layer interact together, with notable changes in the performance of a software system [37]. Iqbal *et al.* [25] found subtle performance bugs that manifest due to the interaction of several layers: configuration options of each software component in a pipeline, configurable low-level libraries that implement functionalities required by different components, the hardware characteristics (*e.g.*, CPU Frequency). The generalizability of performance models is under question, and every variability layer can be a threat to their adoption and usefulness. Overall, performance models of software systems can be challenging to reproduce and reuse as such: there is a need to replicate and adapt performance models to several variability factors.

Reproducible builds. In software development, *Reproducible build* [8] refers to the practice of ensuring that the process of building software yields the same output each time it is performed under identical conditions. It requires that resulting binary artifacts are bit-for-bit identical, regardless of where or when the build process is executed. It enhances transparency, verifiability, and trust in software distribution by enabling anyone to independently verify that the provided software matches the corresponding source code. However, there are various factors that break the reproducible builds property. According to Bajaj *et al.* [4], the build path influences 81% of build reproducibility across a set of Arch Linux and Debian packages. Eliminating this factor resulted in achieving reproducibility. The causes span from build path or timestamps embedded into

the binary to the order in which object files are linked [8, 33]. We can distinguish two approaches to overcome these issues. On the one hand, purely functional package manager such as Nix [13, 14] and Guix [11, 12] perform each build in a dedicated isolated environment. Following this approach up to 93% of Guix packages for x86-64 are bit-by-bit reproducible [20]. Yet, Randrianaina *et al.* [50] show that even for builds performed in an environment with fixed settings, up to 47% of Linux kernel configurations build are unreproducible due configuration options choices. Hence, on the other hand, we have an approach that aims to ensure reproducible builds from within the software, through its configuration. The two approaches are complementary in the sense that they ensure reproducible builds from both the environment in which the build is performed and the software that is built itself.

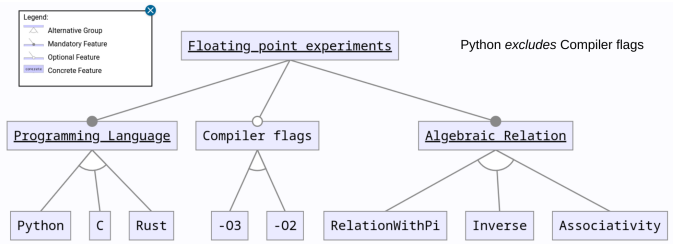

Figure 2: Feature model (excerpt). Inverse (resp. RelationWithPi) corresponds to checking the property $(x*z)/(y*z) = x/y$ (resp. $(x*z*\pi)/(y*z*\pi) = x/y$) with $z, y \neq 0$

4 VISION

Our vision is that there is a need to systematically identify and explicit points of variability that may give rise to reproducibility issues (*e.g.*, language, libraries, compiler, virtual machine, OS, environment variables, *etc.*). Capturing and modelling variability at different layers is a pre-requisite to reason about the configuration space and reproduce some experiments. Furthermore, we aim to embrace this deep variability – by opposition to fixing everything once and for all – since replicating an experiments calls to pro-actively and on purpose different computational settings. On the one hand, reproducing a computational study means running the same computation, and then checking if the results are the same, or at least "close enough" when it comes to *e.g.*, numerical approximations. In practice, we do not test in one run, in one computing environment, with one kind of input data, *etc.* Hence this deep variability can be used to validate an experiment and ensure a certain level of reproducibility. Some configurations are sampled and executed to verify that the results are consistent. On the other hand, scientists and engineers are interested in replicating a scientific study (computational or other): repeating a published protocol, respecting its spirit and intentions but varying the technical details. Deep variability is here a mean to use different software, diversify initial conditions of simulations, and particularly uncover unexpected variations that initially appeared insignificant, aiming to assess the impact on scientific conclusions. Hence this deep variability aims to explore diverse configurations to replicate computational experiments, and gaining more robustness, flexibility, confidence, and eventually consensus of scientific results. It is also a mean to gather new knowledge thanks to the systematic exploration of hypotheses, methodologies, and analyses as encoded in deep variability. In both cases, a model of variability is a key step to represent what can be varied and thus defining a valid envelope of worthy configurations to explore, for either reproducing or replicating a software-intensive experiment.

Running example. To give a very simple example, if we realize that in a C program, $a + (b + c)$ could also have been written as $(a + b) + c$ or even $(c + b) + a$ (associativity property), and that the program is compiled with the option `-O2` but it could also be `-O3`, then we could model these variability points using the well known formalism of feature models [3, 5], as illustrated in Figure 2. For instance, from the feature model of Figure 2 we could select $a + (b + c)$ (Associativity) and `-O3`, and automatically obtain a particular variant of our component, while a choice of $(a + b) + c$) and `-O2` would yield another one. Other variation point such as number of repetitions, number of samples, methods of random generations, types, or simply the programming language can well be specified and then systematically explored.

Then our idea is to resort to statistical reasoning *à la* Monte Carlo to obtain an *uncertainty profile* for software components, along several axes such as accuracy, response time, power consumption or reliability. Monte Carlo methods are a set of approaches that rely on repeated random sampling to obtain numerical results, such as an estimate of the probability distributions that we need here. To extrapolate from observed executions, applied statistics provides a rich set of tools and methodologies that can be adopted for our purposes. But these methods heavily depend on their ability to sample the search space (the *satisfiable* space of a feature model in our case) in such a way that samples are independent and identically distributed. Developing such uniform sampling methods remains a challenge because on the one hand there are numerous constraints among options (*e.g.*, some options are mutually exclusive) and on the other hand the number of options (hence variables) is many orders of magnitude beyond the capabilities of state-of-the-art SAT samplers ($\approx 10^{10.000}$ vs. $\approx 10^{300}$). In the software engineering literature, numerous sampling techniques for testing or measuring programs associated with feature models have been proposed [21, 23, 43, 45, 48, 52]. In practice, there is the need to find operational trade-offs that are close to uniformity with a controlled uncertainty. For instance, the notion of *feature importance* [44] (*e.g.*, the fact that variability factors can be ranked w.r.t. their impact on the performance of a component [40]) be automatically learned or can come from experts' knowledge that would typically prioritize variability of interest as part of the sampling.

Complementary with existing tools and endeavor. There are several tools and initiative to support the idea of reproducible builds. For instance, ReproZip [10] aims to creates a self-contained package for an experiment by automatically tracking and identifying all its required dependencies. Guix [11, 12] and Nix [13, 14] are powerful package and system managers that aim to provide reliable and reproducible build environments. In a sense, ReproZip, Guix and Nix focus on *controlling* and *fixing* the build environment, including variables such as dependencies, compiler versions, and system libraries, aiming to achieve reproducibility. At first glance, this is the "opposite" of what our vision about embracing deep variability

is suggesting, since we aim to vary the build environments. However, and in fact, both approaches are complementary. With the modeling of variability, we capture and document a set of possible and valid configurations for the sake of reproducibility. These configurations can be instantiated and fed to ReproZip, Nix and Guix. Furthermore, we can explore diverse configurations to replicate, and hence expand the generalizability or transferability of results. A variability model will drive the deep exploration while ReproZip, Nix and Guix will be a mean to build, execute, and observe each point of the envelope in a portable and reproducible environment.

REFERENCES

[1] [n. d.]. *Github repository: Reproducibility, associativity, and deep variability.* https://github.com/FAMILIAR-project/reproducibility-associativity

[2] 2021. Hail, software! *Nature Computational Science* 1, 2 (01 Feb 2021), 89–89. https://doi.org/10.1038/s43588-021-00037-8

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer-Verlag.

[4] Rahul Bajaj, Eduardo Fernandes, Bram Adams, and Ahmed E. Hassan. 2023. Unreproducible builds: Time to fix, causes, and correlation with external ecosystem factors. *Empirical Software Engineering* (2023).

[5] David Benavides, Sergio Segura, and Antonio Ruiz-Cortes. 2010. Automated Analysis of Feature Models 20 years Later: a Literature Review. *Information Systems* 35, 6 (2010).

[6] Michael Bentley, Ian Briggs, Ganesh Gopalakrishnan, Dong H Ahn, Ignacio Laguna, Gregory L Lee, and Holger E Jones. 2019. Multi-level analysis of compiler-induced variability and performance tradeoffs. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*. 61–72.

[7] Olivier Bousquet and André Elisseeff. 2002. Stability and generalization. *The Journal of Machine Learning Research* 2 (2002), 499–526.

[8] Reproducible Builds. 2024. Reproducible Builds. https://reproducible-builds.org/ accessed Feb. 2024..

[9] Joshua Carp. 2012. On the plurality of (methodological) worlds: estimating the analytic flexibility of FMRI experiments. *Frontiers in neuroscience* 6 (2012), 149.

[10] Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. 2016. ReproZip: Computational Reproducibility With Ease. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 2085–2088. https://doi.org/10.1145/2882903.2899401

[11] Ludovic Courtès. 2013. Functional Package Management with Guix. In *European Lisp Symposium*. Madrid, Spain. https://inria.hal.science/hal-00824004

[12] Ludovic Courtès and Ricardo Wurmus. 2015. Reproducible and User-Controlled Software Environments in HPC with Guix. In *Euro-Par 2015: Parallel Processing Workshops - Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 9523)*, Sascha Hunold, Alexandru Costan, Domingo Giménez, Alexandru Iosup, Laura Ricci, María Engracia Gómez Requena, Vittorio Scarano, Ana Lucia Varbanescu, Stephen L. Scott, Stefan Lankes, Josef Weidendorfer, and Michael Alexander (Eds.). Springer, 579–591. https://doi.org/10.1007/978-3-319-27308-2_47

[13] Eelco Dolstra. 2006. *The purely functional software deployment model.* Utrecht University.

[14] Eelco Dolstra, Merijn de Jonge, and Eelco Visser. 2004. Nix: A Safe and Policy-Free System for Software Deployment. In *Proceedings of the 18th USENIX Conference on System Administration* (Atlanta, GA) *(LISA '04)*. USENIX Association, USA, 79–92.

[15] R. Döscher, M. Acosta, A. Alessandri, P. Anthoni, T. Arsouze, T. Bergman, R. Bernardello, S. Boussetta, L.-P. Caron, G. Carver, M. Castrillo, F. Catalano, I. Cvijanovic, P. Davini, E. Dekker, F. J. Doblas-Reyes, D. Docquier, P. Echevarria, U. Fladrich, R. Fuentes-Franco, M. Gröger, J. v. Hardenberg, J. Hieronymus, M. P. Karami, J.-P. Keskinen, T. Koenigk, R. Makkonen, F. Massonnet, M. Ménégoz, P. A. Miller, E. Moreno-Chamarro, L. Nieradzik, T. van Noije, P. Nolan, D. O'Donnell, P. Ollinaho, G. van den Oord, P. Ortega, O. T. Prims, A. Ramos, T. Reerink, C. Rousset, Y. Ruprich-Robert, P. Le Sager, T. Schmith, R. Schrödner, F. Serva, V. Sicardi, M. Sloth Madsen, B. Smith, T. Tian, E. Tourigny, P. Uotila, M. Vancoppenolle, S. Wang, D. Wårlind, U. Willén, K. Wyser, S. Yang, X. Yepes-Arbós, and Q. Zhang. 2022. The EC-Earth3 Earth system model for the Coupled Model Intercomparison Project 6. *Geoscientific Model Development* 15, 7 (2022), 2973–3020. https://doi.org/10.5194/gmd-15-2973-2022

[16] Steve M Easterbrook and Timothy C Johns. 2009. Engineering the software for understanding climate change. *Computing in science & engineering* 11, 6 (2009), 65–74.

[17] Naeem Esfahani and Sam Malek. 2013. *Uncertainty in Self-Adaptive Software Systems.* Springer, 214–238.

[18] Tristan Glatard, Lindsay B Lewis, Rafael Ferreira da Silva, Reza Adalat, Natacha Beck, Claude Lepage, Pierre Rioux, Marc-Etienne Rousseau, Tarek Sherif, Ewa Deelman, et al. 2015. Reproducibility of neuroimaging analyses across operating systems. *Frontiers in neuroinformatics* 9 (2015), 12.

[19] Ed HBM Gronenschild, Petra Habets, Heidi IL Jacobs, Ron Mengelers, Nico Rozendaal, Jim Van Os, and Machteld Marcelis. 2012. The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements. *PloS one* 7, 6 (2012), e38234.

[20] GNU Guix. [n. d.]. *Package reproducibility.* https://data.guix.gnu.org/repository/1/branch/master/latest-processed-revision/package-reproducibility

[21] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. 2019. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. *Empir. Softw. Eng.* 24, 2 (2019), 674–717. https://doi.org/10.1007/S10664-018-9635-4

[22] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[23] Ruben Heradio, David Fernandez-Amoros, José A Galindo, David Benavides, and Don Batory. 2022. Uniform and scalable sampling of highly configurable systems. *Empirical Software Engineering* 27, 2 (2022), 1–34.

[24] https://www.wcrp climate.org/. [n. d.]. World climat research program (WCRP).

[25] Md Shahriar Iqbal, Rahul Krishna, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. 2022. Unicorn: reasoning about configurable system performance through the lens of causality. In *Proceedings of the Seventeenth European Conference on Computer Systems* (Rennes, France) *(EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 199–217. https://doi.org/10.1145/3492321.3519575

[26] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 497–508.

[27] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to Sample: Exploiting Similarities Across Environments to Learn Performance Models for Configurable Systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 71–82.

[28] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, and Sven Apel. 2020. The Interplay of Sampling and Machine Learning for Software Performance Prediction. *IEEE Softw.* 37, 4 (2020), 58–66. https://doi.org/10.1109/MS.2020.2987024

[29] Christopher G Knight, Sylvia HE Knight, Neil Massey, Tolu Aina, Carl Christensen, Dave J Frame, Jamie A Kettleborough, Andrew Martin, Stephen Pascoe, Ben Sanderson, et al. 2007. Association of parameter, software, and hardware variation with large-scale behavior across 57,000 climate models. *Proceedings of the National Academy of Sciences* 104, 30 (2007), 12259–12264.

[30] Bran Knowles, Alison Smith-Renner, Forough Poursabzi-Sangdeh, Di Lu, and Halimat Alabi. 2018. Uncertainty in current and future health wearables. *Commun. ACM* 61, 12 (Nov. 2018), 62–67. https://doi.org/10.1145/3199201

[31] Dagmar Krefting, Michael Scheel, Alina Freing, Svenja Specovius, Friedemann Paul, and Alexander Brandt. 2011. Reliability of quantitative neuroimage analysis using freesurfer in distributed environments. In *MICCAI Workshop on High-Performance and Distributed Computing for Medical Imaging.(Toronto, ON)*.

[32] R. Krishna, V. Nair, P. Jamshidi, and T. Menzies. 2020. Whence to Learn? Transferring Knowledge in Configurable Systems using BEETLE. *IEEE Transactions on Software Engineering* (2020), 1–1.

[33] Chris Lamb and Stefano Zacchiroli. 2022. Reproducible Builds: Increasing the Integrity of Software Supply Chains. *IEEE Software* 39, 2 (2022), 62–70. https://doi.org/10.1109/MS.2021.3073045

[34] Dorian Leroy, June Sallou, Johann Bourcier, and Benoit Combemale. 2021. When Scientific Software Meets Software Engineering. *Computer* (2021), 1–11. https://hal.inria.fr/hal-03318348

[35] Luc Lesoil, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2021. The Interaction between Inputs and Configurations fed to Software Systems: an Empirical Study. *CoRR* abs/2112.07279 (2021). arXiv:2112.07279 https://arxiv.org/abs/2112.07279

[36] Luc Lesoil, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2021. Deep Software Variability: Towards Handling Cross-Layer Configuration. In *VaMoS 2021 - 15th International Working Conference on Variability Modelling of Software-Intensive Systems*. Krems / Virtual, Austria, 1–8. https://hal.inria.fr/hal-03084276

[37] Luc Lesoil, Mathieu Acher, Xhevahire Tërnava, Arnaud Blouin, and Jean-Marc Jézéquel. 2021. The Interplay of Compile-time and Run-time Options for Performance Prediction. In *SPLC 2021 - 25th ACM International Systems and Software Product Line Conference - Volume A*. ACM, Leicester, United Kingdom, 1–12. https://doi.org/10.1145/3461001.3471149

[38] Luc Lesoil, Hugo Martin, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2022. Transferring Performance between Distinct Configurable Systems : A Case Study. In *VaMoS '22: 16th International Working Conference on Variability Modelling of Software-Intensive Systems, Florence, Italy, February 23 - 25, 2022*, Paolo Arcaini, Xavier Devroey, and Alessandro Fantechi (Eds.). ACM, 10:1–10:6. https://doi.org/10.1145/3510466.3510486

[39] S. Mahdavi-Hezavehi, P. Avgeriou, and D. Weyns. 2017. A Classification Framework of Uncertainty in Architecture-Based Self-Adaptive Systems With Multiple Quality Requirements. In *Managing Trade-Offs in Adaptable Software Architectures*. Elsevier, 45–77. https://doi.org/10.1016/b978-0-12-802855-1.00003-4

[40] Hugo Martin, Mathieu Acher, Juliana Alves Pereira, Luc Lesoil, Jean-Marc Jézéquel, and Djamel Eddine Khelladi. 2021. Transfer Learning Across Variants and Versions: The Case of Linux Kernel Size. *IEEE Transactions on Software Engineering* (2021), 1–17. https://hal.inria.fr/hal-03358817

[41] François Massonnet, Martin Ménégoz, Mario Acosta, Xavier Yepes-Arbós, Eleftheria Exarchou, and Francisco J Doblas-Reyes. 2020. Replicability of the EC-Earth3 Earth system model under a change in computing environment. *Geoscientific Model Development* 13, 3 (2020), 1165–1178.

[42] Olivier Mesnard and Lorena A. Barba. 2016. Reproducible and replicable CFD: it's harder than you think. arXiv:1605.04339 [physics.comp-ph]

[43] Jeho Oh, Don Batory, and Rubén Heradio. 2023. Finding near-optimal configurations in colossal spaces with statistical guarantees. *ACM Transactions on Software Engineering and Methodology* 33, 1 (2023), 1–36.

[44] Terence Parr, Kerem Turgutlu, Christopher Csiszar, and Jeremy Howard. 2018. Beware Default Random Forest Importances. last access: july 2019.

[45] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, and Jean-Marc Jézéquel. 2020. Sampling Effect on Performance Prediction of Configurable Systems: A Case Study. In *ICPE '20: ACM/SPEC International Conference on Performance Engineering, Edmonton, AB, Canada, April 20-24, 2020*, José Nelson Amaral, Anne Koziolek, Catia Trubiani, and Alexandru Iosup (Eds.). ACM, 277–288. https://doi.org/10.1145/3358960.3379137

[46] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2021. Learning software configuration spaces: A systematic literature review. *J. Syst. Softw.* 182 (2021), 111044. https://doi.org/10.1016/j.jss.2021.111044

[47] Joelle Pineau. ICSE'19 Keynote. Building Reproducible, Reusable, and Robust Machine Learning Software. https://2019.icse-conferences.org/details/icse-2019-Plenary-Sessions/20/Building-Reproducible-Reusable-and-Robust-Machine-Learning-Software.

[48] Quentin Plazar, Mathieu Acher, Gilles Perrouin, Xavier Devroey, and Maxime Cordy. 2019. Uniform Sampling of SAT Solutions for Configurable Systems: Are We There Yet?. In *12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019*. IEEE, 240–251. https://doi.org/10.1109/ICST.2019.00032

[49] R. Botvinik-Nezer et al. 2019. Variability in the analysis of a single neuroimaging dataset by many teams. *bioRxiv* (2019). https://doi.org/10.1101/843193 arXiv:https://www.biorxiv.org/content/early/2019/11/15/843193.full.pdf

[50] Georges Aaron Randrianaina, Djamel Eddine Khelladi, Olivier Zendra, and Mathieu Acher. 2024. Options Matter: Documenting and Fixing Non-Reproducible Builds in Highly-Configurable Systems. In *MSR 2024 - 21th International Conference on Mining Software Repository*. Lisbon, Portugal, 1–11. https://inria.hal.science/hal-04441579

[51] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017*. 39–50. https://doi.org/10.1145/3030207.3030216

[52] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. 2018. A classification of product sampling for software product lines. In *Proceeedings of the 22nd International Conference on Systems and Software Product Line - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*. 1–13. https://doi.org/10.1145/3233027.3233035

[53] Yibo Wang, Ying Wang, Tingwei Zhang, Yue Yu, Shing-Chi Cheung, Hai Yu, and Zhiliang Zhu. 2023. Can Machine Learning Pipelines Be Better Configured?. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (<conf-loc>, <city>San Francisco</city>, <state>CA</state>, <country>USA</country>, </conf-loc>) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 463–475. https://doi.org/10.1145/3611643.3616352

[54] Greg Wilson, Dhavide A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, et al. 2014. Best practices for scientific computing. *PLoS biology* 12, 1 (2014), e1001745.