

# Challenges for Verifying and Validating Scientific Software in Computational Materials Science

Thomas Vogel<sup>\*</sup>, Stephan Druskat<sup>\*†</sup>, Markus Scheidgen<sup>†</sup>, Claudia Draxl<sup>†</sup>, and Lars Grunske<sup>\*</sup>

<sup>\*</sup>Computer Science Department, Humboldt-Universität zu Berlin, Berlin, Germany

<sup>†</sup>German Aerospace Center (DLR), Berlin, Germany

Email: thomas.vogel@informatik.hu-berlin.de, stephan.druskat@dlr.de, grunske@informatik.hu-berlin.de

<sup>†</sup>Physics Department and IRIS Adlershof, Humboldt-Universität zu Berlin, Berlin, Germany

Email: markus.scheidgen@physik.hu-berlin.de, claudia.draxl@physik.hu-berlin.de

**Abstract**—Many fields of science rely on software systems to answer different research questions. For valid results researchers need to trust the results scientific software produces, and consequently quality assurance is of utmost importance. In this paper we are investigating the impact of quality assurance in the domain of computational materials science (CMS). Based on our experience in this domain we formulate challenges for validation and verification of scientific software and their results. Furthermore, we describe directions for future research that can potentially help dealing with these challenges.

**Index Terms**—Verification and Validation, Scientific Software, Computational Materials Science

## I. INTRODUCTION

Software has become an important driver for research in many scientific disciplines such as biology and physics [1]. Scientists often use software in experiments to produce evidence for the validity of their theories, and publish scientific papers based on this evidence [2]. However, in the worst case the validity of such a computational experiment – and thus of the (published) research results – may be jeopardized if the software producing the evidence is not of sufficient quality. A software that has bugs may produce wrong data leading to erroneous evidence. Accordingly, scientific papers have been retracted in the past due to issues with software [3].

Consequently, software engineering principles are being increasingly adopted [4]–[8], and best practices for scientific software development processes have been proposed [9], [10]. At the same time, a clash of cultures between software engineers and domain scientists has been reported [11], [12].

In this context, validation and verification of scientific software are critical, as they establish trust in the software for it to perform the required calculations correctly. In this regard, inadequate behavior of scientific software is a threat to the validity of research results, and has consequently been a main subject of research [13]. To demonstrate the correctness of scientific software, testing is considered essential [14], and has been investigated for scientific software [2], [15]–[19], resulting in tools for testing scientific software [20], [21], the beneficial use of reference data for testing [22], and test-driven development methods [23], [24]. Despite these advances in testing scientific software, all approaches suffer from the *oracle problem* and *large variability* (i.e., a large configuration

space and input domain) of the software under test [19]. Carver et al. [25, p.554] faced the oracle problem in five case studies of computational science and engineering projects, and concluded: “Validation is problematic because it is often difficult, or even impossible, to establish the correct output or result a priori.” In contrast, testing from a software engineering perspective typically considers accurate oracles, that is, the expected output of the software under test is precisely known. This results in a binary oracle: The calculated output either does or does not match the expected output. This contradicts the nature of scientific software, where oracles are unknown or not precisely known. Moreover, the large variability of scientific software poses a challenge to standard testing tools from software engineering because of the large number of tests that are required to comprehensively test the software. Consequently, tests should be well chosen with the goal of allowing scientists to increase their trust in the software [26].

In this paper, we investigate the validation and verification of scientific software in computational materials science (CMS). CMS is concerned with the design and discovery of new materials using computational methods. Based on our experience in the CMS domain, we discuss corresponding challenges such as (i) the oracle problem, and (ii) large configuration spaces of CMS programs, called *codes*, taking the specifics of the domain into account. In the context of the development and use of the NOMAD [27] ecosystem of *codes* and data, we further discuss challenges related to (iii) large-scale, heterogeneous data, and (iv) global software development. Corresponding to these challenges, we proceed to discuss directions for future research on validating and verifying scientific software in CMS.

Throughout the paper, we take the perspective of a CMS scientist who runs calculations to design and analyze materials using a *code* such as **exciting** [28], ABINIT [29], or VASP<sup>1</sup>, or a data-analysis workflow in NOMAD. As the results of a calculation rely on the validity and correctness of the used *code*, our goal is to derive trust levels for *codes* from testing, so that the scientist can increase her trust in the *code*. This paves the way for trustworthy, reproducible calculations and research results.

<sup>1</sup><https://www.vasp.at/>

## II. COMPUTATIONAL MATERIALS SCIENCE

The convergence of theoretical physics and chemistry, materials science and engineering, and computer science into *computational materials science* (CMS) enables the modeling of materials (both existing materials and those that can be created in the future) at the electronic and atomic level. This allows the accurate prediction of how these materials will behave at the microscopic and macroscopic levels, and of their suitability for specific research and commercial applications. CMS is characterized by a healthy, but heterogeneous ecosystem of many different CMS programs, called *codes*, developed by different research groups across the globe. These *codes* are highly domain-specific scientific software packages implementing various theoretical methods. They are executed in high performance computing centers, with millions of CPU hours spent every day, some of them at petascale performance, producing a large stock of equally heterogeneous CMS data.

The NOMAD Center of Excellence<sup>2</sup> (EU/Horizon 2020) aims to enable the CMS community to provide CMS data along the FAIR (findable, accessible, interoperable, and reusable [30]) principles of data sharing. The NOMAD platform provides services that allow scientists to upload raw *code* inputs and outputs and to automatically convert data from all relevant *codes* into a *code*-independent normalized format. It further allows scientists at various levels of expertise to search, inspect, analyze, and visualize all data in this *code*-independent format. Currently, NOMAD supports over 40 *codes*, and stores more than 50 million results of complex calculations regarding properties of materials, including those of the largest US databases, provided by several hundred individual researchers and research groups. Its *code*-independent format uses a hierarchical data schema with over 400 common *code*-independent and almost 2,000 *code*-specific attributes.

The architecture of the NOMAD platform (see Fig. 1), consists of six major components: 1) The *raw data files* Repository where scientists upload, search, and download raw data. 2) *Parsers* and *normalizers* that convert raw data in a *code*-specific format to so-called *Archive* data whose format is *code*-independent. 3) The *Archive* data, that is, the *normalized data* that can be accessed through an API. 4) The *Analytics Toolkit* that allows scientists to apply machine learning techniques to CMS data. 5) The *Encyclopaedia* that aggregates calculations to provide a comprehensive and consistent collection of data for all materials. 6) The *advanced visualization* that uses 3D and virtual-reality techniques to visualize materials at an atomic level.

Following an hour-glass model, the most crucial part of NOMAD is the Archive of normalized data. All supported 40+ *codes* use a different format to represent input and output data for their individual simulations/calculations. *Codes*, data, and data formats differ in the following aspects. First, *codes* implement different methods, with varying computational parameters – and thus, numerical precision – and

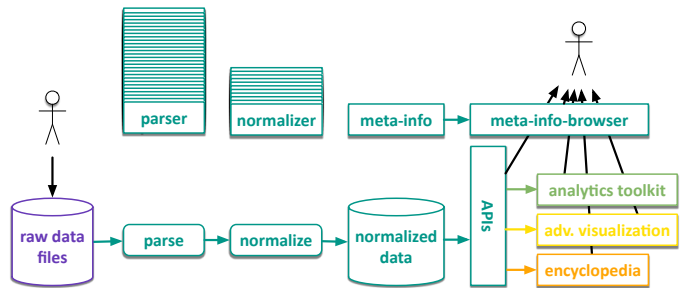


Fig. 1. Architectural view of NOMAD.

individual limitations and trade-offs. Second, *codes* focus on different aspects and produce different physical properties of a simulated material. For instance, a *code* may specialize in electrical, optical, or thermal properties. Third, data is provided in different unit systems (e.g., International System Units (SI) or atomic units). Fourth, although most *codes* use a text format that adheres to some community standards, all quantities are presented in different orders, and matrices and vectors are laid out differently. Quantity values range from strings and dates, simple numerical values, to large vectors, matrices, and tensors of several GB, or even TB, in size. Data formats are not formalized, and documentation is often sparse. Data of individual calculations is often spread over multiple files. Relations between calculations may exist as typically one calculation is based on another. However, such relations are not formalized and have to be deduced from common practices, for instance, a commonly used layout of directories.

To represent data in a normalized and homogeneous form, NOMAD defines an ontology-like data model that unifies all *codes* with a common schema. The schema is used to formalize, categorize, and document all *codes*, as well as *code*-common and *code*-specific quantities, in a single evolving model, called *meta-info*. It uses a proprietary schema language that specializes in describing physical quantities (e.g., with units and vector/matrix dimensions). *meta-info* is independent of distinct technical data formats, and the *Archive* data can be represented in different technical file formats. For example, NOMAD stores the archive data in HDF5, but the API supports access to the data via a JSON representation.

To convert raw CMS data to *Archive* data, NOMAD uses 40+ parsers (one per *code*) and several normalizers. Each set of *code* input/output data is parsed and then processed by all normalizers. Parsers re-produce all quantities found in the raw data in their respective *meta-info* form. Normalizers then compute derived properties, classify simulations, convert units, and relate data with other sources (e.g., external materials databases). In computer language terms, parsers and normalizers only work on a syntactical level, all semantics is added by other NOMAD and potential third-party services.

One of the 40+ *codes* used in the context of NOMAD is **exciting**, a software package implementing density-functional theory (DFT) and many-body perturbation theory [28]. As suggested by its name, **exciting** has a major focus on calculating excited-state properties of materials.

<sup>2</sup><https://nomad-coe.eu/>

### III. PROBLEM STATEMENT AND CHALLENGES

In this section, we first discuss the problem statement, including its relevance to scientific software with a focus on `codes` in computational materials science (CMS). We then proceed to detail challenges in verifying and validating such software.

#### A. Problem Statement

To design and discover new materials, CMS scientists conduct *computational experiments*, in which large-scale, heterogeneous data is processed by data-analysis workflows including `codes`. While certain steps of a workflow are concerned with preparing data (e.g., parsers), the `codes` perform scientific calculations. For an experiment, scientists reuse existing `codes` and data, as well as develop new `codes` that produce new data. Subsequently, they combine all elements in a workflow for execution. The results of such experiments often provide the evidence that the scientists' theories work, which is consequently the basis for scientific papers. A recent example from CMS is the work by Rodrigues Pela et al. [31] who use **exciting** to perform all required calculations.

In this context, there is a multitude of `codes` in CMS implementing a range of theoretical methods. Each of these methods relies on a set of computational parameters that govern the numerical precision of the respective implementation of the method. Choosing the best `code`, and optimal parameters that guarantee high precision is a non-trivial issue, which greatly influences the calculation results. Similar observations of configuration choices impacting research results are reported in bioinformatics [32]. This large configuration space results in high variability, which challenges CMS scientists in predicting how configuration choices impact the results.

Consider, for example, the basic input file for **exciting** in Listing 1. This input file is used for DFT-1/2 calculations (specifically, LDA-1/2) for silicon, in order to compute single-particle band gaps. It first defines the `title` (line 2) and the material `structure`, in this case silicon (lines 3–11). It also features several parameters, such as the presence of `dfthalf` (see line 19), which triggers the DFT-1/2 calculation rather than the default standard DFT. The DFT-1/2 method is configured by the parameters in lines 12–14. Certain parameters have to be determined variationally to obtain optimal results (e.g., `cut`) while others are constant (e.g., `exponent` is usually set to 8).<sup>3</sup>

Listing 1. Example of an input for **exciting**.

```

1 <input>
2 <title>Bulk Silicon: LDA-1/2 example</title>
3 <structure speciespath="SEXCIINGROOT/species">
4   <crystal scale="10.26">
5     <basevect>0.0 0.5 0.5</basevect>
6     <basevect>0.5 0.0 0.5</basevect>
7     <basevect>0.5 0.5 0.0</basevect>
8   </crystal>
9   <species speciesfile="Si.xml" rmt="2.1">
10    <atom coord="0.00 0.00 0.00"></atom>
11    <atom coord="0.25 0.25 0.25"></atom>
12    <dfthalfparam cut="3.90" ampl="1" exponent="8">
13      <shell number="0" ionization="0.25" />
14    </dfthalfparam>
15  </species>

```

<sup>3</sup>For further details, see <http://exciting-code.org/nitrogen-dft05>.

```

16 </structure>
17 <groundstate do="fromscratch" rgkmax="7.0" gmaxvr="14"
18   ngridk="6 6 6" outputlevel="high" xctype="LDA_PW">
19   <dfthalf printV$file="false"/>
20 </groundstate>
21 </input>

```

This (extremely simple) example illustrates only a very small fraction of the complexity of configuring `codes`, as there exist configuration parameters that are mutable or constant, as well as dependencies between parameters (e.g., only if the `dfthalf` method is selected, it can be further configured by the parameters in `dfthalfparam`).

Consequently, given the complexity of configuring `codes` for their use, and of initially developing such `codes`, scientists may generally ask themselves two questions:

- (1) *Is the theoretical method I have developed valid?*, or in the case of reusing an existing method: *Have I selected and configured appropriately the theoretical method and therefore, the code implementing this method?*
- (2) *Is the code implementing this method correct?*

While the first question concerns the *validation* of software ("Am I building the right product?"), the second one addresses its *verification* ("Am I building the product right", cf. [33]). Thus, validation is about the adequate (proper use), and verification about the correct (absence of bugs), functional behavior of software such as the `codes` in CMS. Hence, both validation and verification of CMS `codes` are required to obtain trustworthy results from calculations. Otherwise, the `codes` pose a potential major threat to the validity of the experiments and research results, as any inadequate or incorrect `code` refutes these results. However, validating and verifying `codes` in computational materials science poses challenges, which we will discuss in the next section.

#### B. Challenges for Validating and Verifying CMS Codes

In the following, we discuss major challenges for the verification and validation of scientific software, which - based on our experience in (research) software engineering - are caused by the described factors: experimental nature and complexity of `codes`, and complexity of the data processed by them.

1) *Lack of Precise Oracles*: As scientists use computational calculations to explore new ideas and theoretical methods, the outcome of a calculation is generally not known at all, or at least not precisely known a priori [4], [25], [34]. Other reasons for this are the complexity of the calculations, and the fact that the calculations may return a range of different answers, which makes it difficult for scientists to predict the outcome [19].

This causes uncertainty about calculation results, as there is no precise notion of their correctness. Consequently, the same is true for the software used to explore such new ideas and methods, which prohibits precise oracles to be defined for quality assurance techniques such as software testing.

Consider, for example, a CMS scientist who performs one of the following calculations. She simulates existing materials with well-known properties on a new implementation (`code`) of an existing, altered, or completely new method, or she simulates unknown materials on an existing `code` of established

methods, or even does so in bulk to explore the huge space of unknown potential materials. To be more specific, considering the example presented previously (see Listing 1), estimating the impact of varying input parameters such as `cut` on the results may be difficult. In all cases, the expected result of the calculation is not known and cannot be predicted a priori.

In contrast, the result obtained for a specific property of an input material is *likely* to be correct if the calculated property value is statistically similar to that of other well-known materials of the same class, assuming we can classify the material. Outliers, in turn, may indicate one of the following cases:

- (i) Discovery of a highly interesting material. In this positive case, a material has been discovered, whose properties are different from existing materials of the same class.
- (ii) Faulty theoretical method and/or parameters. In this negative case, a scientist either made a mistake when developing a new method, which manifests in its implementation (`code`), or used a nonsensical combination of method and parameters when using an existing method and `code`. Here, the `code` is, or may be, free of bugs.
- (iii) A bug in the software (`code`). This is the other negative case, in which the `code` contains a bug that caused the faulty results. More specifically, the `code` is a faulty implementation of a valid theoretical method.

This perspective gives rise to developing *statistical oracles*, which judge the plausibility of computational results and provide corresponding feedback to scientists, which in turn establishes confidence in these results. Non-plausible results, which are expected to be rare, need to be inspected manually and classified according to the cases (i), (ii), and (iii). The use of such statistical oracles is conceivable in quality assurance techniques such as software testing. However, from a software engineering point of view, testing mainly focuses on precise oracles and assertions, so that state-of-the-art and state-of-the-practice testing approaches, or even test-driven development, cannot be directly applied here. Consequently, quality assurance techniques such as systematic testing known from software engineering [4], [10] are rarely adopted for scientific software [19]. Therefore, increasing the confidence of scientists in computational results requires quality assurance techniques which can be applied to scientific software packages *a-posteriori* and in an automated manner [10].

This results in the following challenges for leveraging statistical oracles in testing of scientific software:

- Which methods and techniques shall be used to provide a statistical oracle?
- How can such methods reliably judge the success and potential failure of a set of executed tests?

2) *Large Configuration Space*: As discussed above, the experimental nature of scientific software (`codes`) typically results in a large configuration space. This comprises the selection of algorithms provided by a software, as well as fine-tuning the selected algorithm through parameters, which in turn results in high variability and a large number of options

for executing the software [35], [36]. At the same time, the choice of configuration influences the calculation results [32].

An example from CMS is the calculation of single-particle band gaps, for which **exciting** can be customized to perform a calculation that is further configured by a set of parameters (see Section III-A). In the context of NOMAD, 40+ `codes` such as **exciting** are used, which multiplies the variability that CMS scientists have to cope with.

This variability challenges scientists to select and configure appropriate `codes` for calculations. As the selection and configuration of `codes` can greatly influence the calculation results, CMS scientists should be supported *a-priori* and in an automated manner during this process. Such support should guide scientists in implementing a method to prevent the introduction of basic faults, before a calculation is conducted. It therefore promotes the validation of the configured methods/`codes` and of the conducted calculations. In CMS, such support may suggest to scientists the use of trusted `codes` and methods (including parameters) for specific materials and/or properties that are of interest for a specific calculation. For example, a recommendation may be to use an all-electron `code` and a self-interaction corrected exchange-correlation function to properly account for electron-electron interactions for a heavy material like cerium.

Moreover, the large configuration space and the corresponding variability of `codes` also challenges the validation and verification of these `codes` through testing, in that it is infeasible to test *all* possible configurations. The large configuration space impedes manual identification of test cases and thereby of configurations to be tested (*cf.* [19], [35], [36]). Thus, an automated sampling of the configuration space to identify representative configurations to be tested is required. In general, this constitutes a combinatorial interaction testing problem [37] while a solution for this problem has to be tailored to the CMS domain. Consequently, coping with the large configuration space requires automated support for scientists in using `codes`, as well as intelligent testing techniques, which account for the following challenges:

- What are appropriate sampling strategies for selecting a subset of scientific computations (i.e., a combination of `code`, `code` configuration, and input data in CMS) that are likely to reveal a failure in a scientific software?
- How to exploit results of previous calculations and test runs of `codes`, to automatically determine the required support for scientists? Particularly, how to exploit statistical information from automated testing, to suggest methods and corresponding `codes` (including configuration parameters) to scientists for a specific calculation?

3) *Large-Scale, Heterogeneous Data*: Scientific software often processes large-scale, heterogeneous data, e.g., in climate research [38], and in CMS [27] where software operates on data up to several TB in size and encoded in different `code`-specific formats that are mostly neither formalized nor well-documented. Thus, calculations using results of multiple `codes` in NOMAD require pre- and post-processing steps to transform input/output data between the normalized Archive

format (cf. Section II) and the code-specific formats, to integrate machine learning, or for visualization. For instance, parsers (one for each code-specific format) and normalizers are used to translate code-specific input/output data of a code for storage in the Archive and future use. Consequently, codes implementing theoretical methods are embedded in a workflow, together with programs implementing such pre- and post-processing steps. One workflow example is a machine learning approach applied to properties computed by multiple codes over many materials, to find predictors for a specific materials property. Here, the properties have to be computed, parsed, and normalized for the learning.

Consequently, validation and verification have to address the whole workflow. Otherwise, a faulty pre- or post-processing step might introduce faults into the data causing either wrong calculations by the (bug-free) codes, or wrong presentations and interpretations of the results by scientists. Hence, a fault might be located in any part of a data-analysis workflow (cf. NOMAD workflow in Section II).

Thus, the selection of test data (including the pre- and post-processing steps) for testing workflows is crucial. For instance, considering a workflow that classifies materials based on their electrical resistivity and conductivity, tests should cover calculation data from different codes implying different methods, unit systems, respective parser and normalizer chains, as well as representatives from different classes of materials (e.g., super-, semi-, non-, conductors).

This heterogeneity, together with the scale of the data, results in high variability at the data level (in addition to the variability of the codes discussed in the context of large configuration spaces), which challenges the validation and verification of data-analysis workflows:

- How to identify and sample valid/realistic test data for codes and workflows that likely reveal a failure?
- How to improve the quality of the pre-/post-processing steps that handle large-scale, heterogeneous data?

4) *Global Software Development*: In CMS, scientists across the globe explore theoretical methods and develop codes. An ecosystem of several hundred scientists and research groups has emerged around NOMAD, fostering reuse of data for new and reproducing calculations, reuse of codes in workflows, and development of new codes based on existing ones.

However, reuse is often kept implicit, e.g., for lack of common workflow descriptions [39]. For instance, relations between calculations do exist, but such relations often have to be deduced from common practices such as a commonly used layout of directories. For example, to derive elasticity properties for a material with **exciting**, a series of simulations with varying forces acting on the simulated material has to be performed [40]. Only an analysis of all these simulations allows scientists to derive the desired elastic constants. However, the intent behind the series of simulations is not always formalized. From the perspective of NOMAD, or data reuse in general, the parameter study's relations between those simulations and their underlying intent have to be deduced. Even originally unrelated simulations from different codes

could be used in a parameter study, provided one identifies respective data based on comparable methods and parameters.

Moreover, codes are sometimes not well-documented – with regards to any and all levels of documentation, e.g., requirements, system modeling, architectural design, maintenance guidelines, and user documentation – or no longer maintained, their data format may not be formalized, and the corresponding parser may only produce partial parses of the format. Finally, the quality of a code might be unknown or the quality might differ, depending on the degree to which quality assurance techniques such as testing are adopted. These aspects are caused by general issues of global software development concerning knowledge, project, and process management [41], and they challenge the validation and verification of codes that are (re)used by scientists other than the scientists developing the codes:

- How to validate and verify third-party code that is not well-documented, not sufficiently tested, and whose data format is not formalized? How to achieve trustworthy workflows that use data from different sources in different codes?
- How to extract and mine relations between calculations to leverage integration testing and to generally improve quality/trust levels of codes, for instance, by externalizing assurances obtained for reused codes?

#### IV. DIRECTIONS FOR FUTURE RESEARCH

1) *Lack of Precise Oracles*: Currently, the confidence about scientific research results in the CMS domain is addressed by scientific workflow systems using the notion of *provenance*, since all executions of codes and the corresponding input and output data are documented in NOMAD<sup>4</sup>. A recent community effort [42] has, for the first time, assessed and compared the quality of DFT results computed by several codes for a set of materials. More recently, the effect of computational parameters have been systematically assessed, involving four different codes [43]. The goal here is to automate the collection of workflow metadata to enable reproducibility of scientific results.

Based on the available NOMAD data, the next step will be to define a notion of a *statistical oracle*, which uses statistical methods for identifying the correctness of a computational calculation [44]. Unlike usual oracles used in software testing such as oracles derived from requirement specifications or models, gold standard oracles, or human oracles, the decision of a statistical oracle as envisioned is, by definition, not always correct. To apply statistical methods, results in the neighborhood of the computational calculation need to be investigated. Chan et al. [45] provide a general algorithm based on mesh specifications and machine learning for this problem. However, defining the neighborhood in CMS requires looking at the used materials, codes including its parameters, and computational environment. Furthermore, the selection of appropriate heuristics, which keep the oracle's failure at a

<sup>4</sup>[https://metainfo.nomad-coe.eu/nomadmetainfo\\_public/archive.html](https://metainfo.nomad-coe.eu/nomadmetainfo_public/archive.html)

minimum, is an open problem that has been little researched in general [44] and needs to be tailored to data-driven CMS. Finally, if the neighborhood of a calculation is not available in NOMAD, specific computational calculations can be provided by mutation sensitivity testing [46] and modeling as well as approximation techniques of the input space [36].

Beyond this, it will be very interesting to apply the concept of metamorphic testing [47] that is specifically designed to test software without an oracle [48], [49]. The idea is to identify and refine a set of metamorphic relations between the software inputs and outputs. Just to give an abstract example, for a square root function  $\text{sqrt}(x)$  the relation  $x = \text{sqrt}(x) * \text{sqrt}(x)$  should hold under reasonable floating point accuracy assumptions. Identifying such relations is highly domain dependent. However, automatic techniques based on machine learning have been proposed [44], [50]–[52] and successfully applied to the bio-medical [53] and particle physics [54] domains. Transferring the concept of metamorphic testing requires domain expertise since the identified relations need to be understood and explained. The explainability of the relations is a primary challenge. However, it is also a significant opportunity for the CMS community since the scientists might learn hidden relations from their `codes` which were previously unknown. This may strengthen the understanding and help to refine the underlying theories.

2) *Large Configuration Space*: `Codes` in CMS are used by selecting a desired method and by fine-tuning this method through a set of parameters. Thus, there is not *the* perfect implementation, but each `code` is actually a tool box that can be instantiated in a huge number of variants. In sum, this leads to a combinatorial explosion of possible computations, and testing all of them is infeasible. Instead, appropriate sampling strategies are required which are effective and efficient at the same time. Possible scenarios for first tests are, for instance, to stay within the same `code` family and vary, for a given method, the parameter space; or select the best possible (fully converged) calculations from different `codes`.

The steps for future research to deal with the large configuration space when verifying and validating CMS `codes` and calculations require effective methods for configuration space sampling and automated test input generation. Concerning the sampling of suitable input data for generated test cases, one idea [35] is to apply combinatorial testing and test case selection techniques, which have been exploited in software product line (SPL) engineering [55], [56]. The goal of these techniques is to select a promising subset of product variants when testing all variants of an SPL is infeasible. However, SPL engineering focuses on testing interactions of features, which may be present in a product variant, or not. In contrast, the configuration space of CMS software comprises a set of non-boolean parameters, which demand different coverage metrics and sampling strategies. Furthermore, our hypothesis is that the size of the configuration space exceeds the configuration space of very large existing SPLs such as the Linux kernel. As a result, we have to enrich the sampling strategies with modeling techniques for the input space as

proposed by Vilkomir et al. [36]. Another direction for future research is to use a recommender system exploiting statistical information obtained from existing computational calculations in the NOMAD Repository. Passing and non-passing test cases will be classified statistically to provide valuable information about the adequacy of `code` configurations. Our aim is to exploit this information to derive a recommender system that assists scientists in configuring `codes` for their specific needs.

3) *Large-Scale, Heterogeneous Data*: There are two aspects of the data diversity problem in CMS. First, we have different representations of the same information, for instance, different file formats, layouts of matrices, units, etc. Second, `codes` provide different kinds of information, for instance, `codes` specializing in electronic properties vs. `codes` specializing in thermal properties. The former problem can be solved by finding the right abstractions, the latter by defining relations between properties (e.g., identifying generalizations, categories of properties, or associations). Both aspects can be tackled by formal data models.

Modeling data has a long history in computer science, and has different methods in different *technical spaces* [57] such as schemas for data exchange (XML, JSON), relational algebra in databases, ontologies in semantic web, or formal grammars and meta-models in computer languages. Applications often require transforming data from a representation in one space to a representation in another (e.g., reading data from a database organized in tables and sending it over the internet in hierarchically nested JSON format). The scale of the data increases the problem since specialized technologies have to be combined. For instance, search engines, distributed computing platforms, and nosql-databases have to work hand in hand. Each technology potentially requires its own specialized data representation. To cope with this, data must be modeled at a level that is independent from concrete technical spaces.

The CMS domain (or scientific software in general) presents a further challenge, since most existing methods for formally defining data types fall short as they neglect the nature of scientific data and offer no or insufficient support for vectors, matrices, tensors, their dimensions, and units. Therefore, NOMAD defines its own schema language *meta-info* [58] that is independent of the concrete data representation (e.g., text files, HDF5 files, or databases). In all its representations, data retains its inherent structure and types as defined in meta-info. Furthermore, meta-info categorizes properties into sections, and defines relations between properties and their categories. Some of the meta-info is common and shared by many `codes`, some definitions are `code` specific.

This formal model of CMS data can foster the quality of `codes` and workflows in several ways, e.g., by applying methods from model-based testing. First, a formal model can support generating realistic large-scale test data and asserting test coverage with respect to the input of `codes`. Secondly, it is a formal definition of the possible data space. Constraints defined at the meta-info level can be used to automatically assert the plausibility of calculated properties. Finally, it can automatize the development of mappings between technical

spaces (i.e., parsers and normalizers) by declaratively defining mappings, from which operational transformations are automatically derived. This avoids error-prone manual implementations of parsers and normalizers.

4) *Global Software Development*: Global software development challenges to the verification and validation of `codes` (re)used in CMS studies pertain mainly to two factors: (i) The large and diverse development ecosystem, which produces `codes` that differ in quality (e.g., levels of documentation and testing); (ii) the lack of explication of intent when combining multiple calculations and `codes` in workflows.

Efforts to consolidate the diversity of the ecosystem in terms of software quality will have to be implemented as community processes. Code development should adopt best practices of software engineering [10], [59]. These practices must be adapted to the needs of CMS, for instance, in regard to testing (*cf.* Section IV-1). Similar efforts have been made in astronomy<sup>5</sup>. Such efforts will ease the integration and testing of `codes` developed by other scientists in workflows.

Despite the use of workflow systems in CMS (*cf.* Section IV-1), metadata explicating the intent behind a parametrization and combination of calculations/`codes` within a single study is often missing. Therefore, any intent can only be deduced from potentially interrelated, non-formalized information such as directory structures. To explicate intent, future efforts should develop and apply requirements for formalized metadata, for instance, by using the Common Workflow Language [39], a specification for portable and scalable workflow descriptions with dedicated metadata. Similarly to efforts regarding the development ecosystem as such, this must be achieved through a standardization process within the CMS community. Additionally, automatic methods for the discovery of intentional process models based on Hidden Markov Models [60], [61] can be adapted to mine implicit relations between calculations/`codes`. These models can guide the integration testing of data analysis workflows.

## V. CONCLUSIONS

In this paper we discussed challenges for the validation and verification of scientific software in computational materials science (CMS). We conclude that most of the problems are similar to other domains [14], [16], [18], [21], [23], [26], [44], [62], [63] and solution principles derived for the CMS domain might be generalizable to other domains. However, the effort of the CMS community to provide results of their computational experiments in the NOMAD Repository [27] based on the FAIR principle [30] provides a significant opportunity for fundamental research on validation and verification of scientific software. For instance, based on the NOMAD data, novel strategies to tackle the oracle problem, can be developed. With this research, we envision trust levels for `codes` so that scientists increase their trust in `codes` to obtain trustworthy, reproducible calculations and research results.

<sup>5</sup><https://eas.unige.ch/EWASS2017/session.jsp?id=SS16>

## REFERENCES

- [1] J. C. Carver, N. P. C. Hong, and G. K. Thiruvathukal, *Software Engineering for Science*. CRC Press, 2016.
- [2] R. Sanders and D. Kelly, "Dealing with risk in scientific software development," *IEEE Software*, vol. 25, no. 4, pp. 21–28, 2008.
- [3] G. Miller, "A scientist's nightmare: Software problem leads to five retractions," *Science*, vol. 314, no. 5807, pp. 1856–1857, 2006.
- [4] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?" in *ICSE Workshop on Software Engineering for Computational Science and Engineering*, ser. SECSE. IEEE, 2009, pp. 1–8.
- [5] L. Nguyen-Hoan, S. Flint, and R. Sankaranarayana, "A survey of scientific software development," in *International Symposium on Empirical Software Engineering and Measurement*. ACM, 2010, pp. 12:1–12:10.
- [6] D. Heaton and J. C. Carver, "Claims about the use of software engineering practices in science: A systematic literature review," *Information & Software Technology*, vol. 67, pp. 207–219, 2015.
- [7] T. Storer, "Bridging the chasm: A survey of software engineering practice in scientific programming," *ACM Comput. Surv.*, vol. 50, no. 4, pp. 47:1–47:32, 2017.
- [8] A. N. Johanson and W. Hasselbring, "Software engineering for computational science: Past, present, future," *Computing in Science and Engineering*, vol. 20, no. 2, pp. 90–109, 2018.
- [9] D. Kelly, D. Hook, and R. Sanders, "Five recommended practices for computational scientists who write software," *Computing in Science and Engineering*, vol. 11, no. 5, pp. 48–53, 2009.
- [10] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson, "Best practices for scientific computing," *PLOS Biology*, vol. 12, no. 1, pp. 1–7, 01 2014.
- [11] J. Segal, "When software engineers met research scientists: A case study," *Empirical Software Eng.*, vol. 10, no. 4, pp. 517–536, 2005.
- [12] —, "Scientists and software engineers: A tale of two cultures," in *Workshop of the Psychology of Programming Interest Group*, 2008, p. 6.
- [13] L. Hatton and A. Roberts, "How accurate is scientific software?" *IEEE Trans. Software Eng.*, vol. 20, no. 10, pp. 785–797, 1994.
- [14] T. Clune, M. Rilee, and D. Rouson, "Testing as an essential process for developing and maintaining scientific software," in *The 2nd Workshop on Sustainable Software for Science: Practices and Experiences*, 2014.
- [15] D. Kelly, R. Gray, and Y. Shao, "Examining random and designed tests to detect code mistakes in scientific software," *J. Comput. Science*, vol. 2, no. 1, pp. 47–56, 2011.
- [16] D. Kelly, S. Thorsteinson, and D. Hook, "Scientific software testing: Analysis with four dimensions," *Softw.*, vol. 28, no. 3, pp. 84–90, 2011.
- [17] P. C. R. Lane and F. Gobet, "A theory-driven testing methodology for developing scientific software," *J. Exp. Theor. Artif. Intell.*, vol. 24, no. 4, pp. 421–456, 2012.
- [18] D. Kelly, "Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software," *Journal of Systems and Software*, vol. 109, pp. 50–61, 2015.
- [19] U. Kanewala and J. M. Bieman, "Testing scientific software: A systematic literature review," *Information and Software Technology*, vol. 56, no. 10, pp. 1219–1232, 2014.
- [20] M. C. Smith, R. L. Kelsey, J. M. Riese, and G. A. Young, "Creating a flexible environment for testing scientific software," in *Enabling Technologies for Simulation Science VIII, SPIE 5423*, 2004, pp. 288–296.
- [21] P. F. Dubois, "Testing scientific programs," *Computing in Science and Engineering*, vol. 14, no. 4, pp. 69–73, 2012.
- [22] M. Cox and P. Harris, "Design and use of reference data sets for testing scientific software," *Analytica Chimica Acta*, vol. 380, no. 2, pp. 339–351, 1999.
- [23] A. Nanthaamornphong and J. C. Carver, "Test-driven development in scientific software: a survey," *Software Quality Journal*, vol. 25, no. 2, pp. 343–372, Jun 2017.
- [24] T. L. Clune and R. Rood, "Software testing and verification in model development," *IEEE Software*, vol. 28, no. 6, pp. 49–55, 2011.
- [25] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post, "Software development environments for scientific and engineering software: A series of case studies," in *29th International Conference on Software Engineering (ICSE)*. IEEE, 2007, pp. 550–559.
- [26] D. Hook and D. Kelly, "Testing for trustworthiness in scientific software," in *ICSE Workshop on Software Engineering for Computational Science and Engineering, SE-CSE*. IEEE, 2009, pp. 59–64.

- [27] C. Draxl and M. Scheffler, "Nomad: The fair concept for big data-driven materials science," *MRS Bulletin*, vol. 43, no. 9, pp. 676–682, 2018.
- [28] A. Gulans, S. Kontur, C. Meisenbichler, D. Nabok, P. Pavone, S. Rigamonti, S. Sagmeister, U. Werner, and C. Draxl, "exciting: a full-potential all-electron package implementing density-functional theory and many-body perturbation theory," *Journal of Physics: Condensed Matter*, vol. 26, no. 36, p. 363202, 2014.
- [29] X. Gonze, F. Jollet, F. Abreu Araujo, D. Adams, B. Amadon, T. Applencourt, C. Audouze, J. M. Beuken, J. Bieder, A. Bokhanchuk, E. Bousquet, F. Bruneval, D. Caliste, M. Côté, F. Dahm, F. Da Pieve, M. Delaveau, M. Di Gennaro, B. Dorado, C. Espejo, G. Geneste, L. Genovese, A. Gerossier, M. Giantomassi, Y. Gillet, D. R. Hamann, L. He, G. Jomard, J. Laflamme Janssen, S. Le Roux, A. Levitt, A. Lherbier, F. Liu, I. Lukačević, A. Martin, C. Martins, M. J. T. Oliveira, S. Poncé, Y. Pouillon, T. Rangel, G. M. Rignanese, A. H. Romero, B. Rousseau, O. Rubel, A. A. Shukri, M. Stankovski, M. Torrent, M. J. Van Setten, B. Van Troeye, M. J. Verstraete, D. Waroquiers, J. Wiktor, B. Xu, A. Zhou, and J. W. Zwanziger, "Recent developments in the ABINIT software package," *Computer Physics Communications*, vol. 205, pp. 106–131, Aug. 2016.
- [30] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. G. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. C. 't Hoen, R. Hoof, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, p. 160018, Mar. 2016.
- [31] R. Rodrigues Pela, A. Gulans, and C. Draxl, "The lda-1/2 method applied to atoms and molecules," *Journal of Chemical Theory and Computation*, vol. 14, no. 9, pp. 4678–4686, 2018, PMID: 30119607.
- [32] M. Cashman, M. B. Cohen, P. Ranjan, and R. W. Cottingham, "Navigating the maze: The impact of configurability in bioinformatics software," in *33rd International Conference on Automated Software Engineering*, ser. ASE. ACM, 2018, pp. 757–767.
- [33] B. W. Boehm, "Verifying and validating software requirements and design specifications," *IEEE Software*, vol. 1, no. 1, pp. 75–88, 1984.
- [34] D. Kelly, S. Smith, and N. Meng, "Software engineering for scientists," *Computing in Science Engineering*, vol. 13, no. 5, pp. 7–11, 2011.
- [35] H. Remmel, B. Paech, P. Bastian, and C. Engwer, "System testing a scientific framework using a regression-test environment," *Computing in Science and Engineering*, vol. 14, no. 2, pp. 38–45, 2012.
- [36] S. A. Vilkomir, W. T. Swain, J. H. Poore, and K. T. Clarno, "Modeling input space for testing scientific computational software: A case study," in *8th International Conference on Computational Science (ICCS)*, ser. LNCS, vol. 5103. Springer, 2008, pp. 291–300.
- [37] C. Yilmaz, S. Fouch, M. B. Cohen, A. Porter, G. Demiroz, and U. Koc, "Moving forward with combinatorial interaction testing," *Computer*, vol. 47, no. 2, pp. 37–45, 2014.
- [38] S. M. Easterbrook, "Climate change: A grand software challenge," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10. ACM, 2010, pp. 99–104.
- [39] B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, and L. Stojanovic, "Common Workflow Language, v1.0," Common Workflow Language working group, Specification, Jul. 2016. [Online]. Available: <https://w3id.org/cwl/v1.0/>
- [40] R. Golesorkhtabar, P. Pavone, J. Spitaler, P. Puschnig, and C. Draxl, "Elastic: A tool for calculating second-order elastic constants from first principles," *Computer Physics Communications*, vol. 184, no. 8, pp. 1861–1873, 2013.
- [41] J. D. Herbsleb and D. Moitra, "Global software development," *IEEE Software*, vol. 18, no. 2, pp. 16–20, March 2001.
- [42] K. Lejaeghere, G. Bihlmayer, T. Björkman, P. Blaha, S. Blügel, V. Blum, D. Caliste, I. E. Castelli, S. J. Clark, A. Dal Corso, S. de Gironcoli, T. Deutsch, J. K. Dewhurst, I. Di Marco, C. Draxl, M. Dułak, O. Eriksson, J. A. Flores-Livas, K. F. Garrity, L. Genovese, P. Giannozzi, M. Giantomassi, S. Goedecker, X. Gonze, O. Grånäs, E. K. U. Gross, A. Gulans, F. Gygi, D. R. Hamann, P. J. Hasnip, N. A. W. Holzwarth, D. Iuşan, D. B. Jochym, F. Jollet, D. Jones, G. Kresse, K. Koepfner, E. Küçükbenli, Y. O. Kvashnin, I. L. M. Locht, S. Lubeck, M. Marsman, N. Marzari, U. Nitzsche, L. Nordström, T. Ozaki, L. Paulatto, C. J. Pickard, W. Poelmans, M. I. J. Probert, K. Refson, M. Richter, G.-M. Rignanese, S. Saha, M. Scheffler, M. Schlipf, K. Schwarz, S. Sharma, F. Tavazza, P. Thunström, A. Tkatchenko, M. Torrent, D. Vanderbilt, M. J. van Setten, V. Van Speybroeck, J. M. Wills, J. R. Yates, G.-X. Zhang, and S. Cottenier, "Reproducibility in density functional theory calculations of solids," *Science*, vol. 351, no. 6280, 2016.
- [43] C. Carbogno, K. Thygesen, B. Bieniek, C. Draxl, L. Ghiringhelli, A. Gulans, O. Hofmann, K. Jacobsen, S. Lubeck, J. Mortensen, M. Strange, E. Wruss, and M. Scheffler, "Quality Control of Numerical Settings for DFT Calculations and Materials Databases," in *APS Meeting Abstracts*, 2018, p. P12.003.
- [44] U. Kanewala and J. M. Bieman, "Techniques for testing scientific programs without an oracle," in *Intl. Workshop on Software Engineering for Computational Science and Engineering*. IEEE, 2013, pp. 48–57.
- [45] W. K. Chan, S. Cheung, J. C. F. Ho, and T. H. Tse, "PAT: A pattern classification approach to automatic reference oracles for the testing of mesh simplification programs," *Journal of Systems and Software*, vol. 82, no. 3, pp. 422–434, 2009.
- [46] D. Hook and D. Kelly, "Mutation sensitivity testing," *Computing in Science and Engineering*, vol. 11, no. 6, pp. 40–47, 2009.
- [47] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," HKUST-CS98-01, Department of Computer Science, Hong Kong, Tech. Rep., 1998.
- [48] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Trans. Software Eng.*, vol. 41, no. 5, pp. 507–525, 2015.
- [49] S. Segura, G. Fraser, A. B. Sánchez, and A. R. Cortés, "A survey on metamorphic testing," *IEEE Trans. Software Eng.*, vol. 42, no. 9, pp. 805–824, 2016.
- [50] A. Gotlieb and B. Botella, "Automated metamorphic testing," in *27th International Computer Software and Applications Conference (COMPSAC)*. IEEE, 2003, pp. 34–40.
- [51] C. Murphy, G. E. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in *Intl. Conf. on Softw. Engineering & Knowledge Engineering*. KSI, 2008, pp. 867–872.
- [52] X. Xie, J. W. K. Ho, C. Murphy, G. E. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Sys. and Softw.*, vol. 84, no. 4, pp. 544–558, 2011.
- [53] T. Y. Chen, J. W. Ho, H. Liu, and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing," *BMC bioinformatics*, vol. 10, no. 1, p. 24, 2009.
- [54] J. Ding and D. Zhang, "A machine learning approach for developing test oracles for testing scientific software," in *Intl. Conference on Software Engineering and Knowledge Engineering*. KSI, 2016, pp. 390–395.
- [55] A. Sarkar, J. Guo, N. Siegmund, S. Apel, and K. Czarnecki, "Cost-efficient sampling for performance prediction of configurable systems (T)," in *30th International Conference on Automated Software Engineering, ASE*. IEEE, 2015, pp. 342–352.
- [56] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake, "A classification and survey of analysis strategies for software product lines," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 6:1–6:45, 2014.
- [57] I. Ivanov, J. Bézivin, and M. Aksit, "Technological spaces: An initial appraisal," 10 2002, pp. 1–6.
- [58] L. M. Ghiringhelli, C. Carbogno, S. Levchenko, F. Mohamed, G. Huhs, M. Lueders, M. Oliveira, and M. Scheffler, "Towards a common format for computational material science data," 2016.
- [59] J. Hastings, K. Haug, and C. Steinbeck, "Ten recommendations for software engineering in research," *GigaScience*, vol. 3, Dec. 2014.
- [60] G. Khodabandelou, C. Hug, R. Deneckère, and C. Salinesi, "Supervised intentional process models discovery using Hidden Markov models," in *7th International Conference on Research Challenges in Information Science (RCIS)*, 2013, pp. 1–11.
- [61] —, "Unsupervised Discovery of Intentional Process Models from Event Logs," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. ACM, 2014, pp. 282–291.
- [62] C. Morris and J. Segal, "Some challenges facing scientific software developers: The case of molecular biology," in *Fifth International Conference on e-Science*. IEEE, 2009, pp. 216–222.
- [63] W. A. Reupke, E. Srinivasan, P. V. Rigerink, and D. N. Card, "The need for a rigorous development and testing methodology for medical software," in *First Annual IEEE Symposium on Computer-Based Medical Systems (CBMS'88)*. IEEE, 1988, pp. 15–20.