Scientific Workflow Management for Software Quality Assessment Replication: An Open Source Architecture

José Pereira dos Reis[1,2](✉), Fernando Brito e Abreu[2], Glauco de F. Carneiro[3], and Duarte Almeida[2]

[1] Instituto Superior de Tecnologias Avançadas (ISTEC), Lisboa, Portugal
josevicente.reis@my.istec.pt

[2] Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL, Lisboa, Portugal
{jvprs,fba,dsbaa}@iscte-iul.pt

[3] Federal University of Sergipe (UFS), Aracaju, Brazil
glauco.carneiro@dcomp.ufs.br

Abstract. Replication of research experiments is important for establishing the validity and generalizability of findings, building a cumulative body of knowledge, and addressing issues of publication bias. The quest for replication led to the concept of scientific workflow, a structured and systematic process for carrying out research that defines a series of steps, methods, and tools needed to collect and analyze data, and generate results.

In this study, we propose a cloud-based framework built upon open source software, which facilitates the construction and execution of workflows for the replication/reproduction of software quality studies. To demonstrate its feasibility, we describe the replication of a software quality experiment on automatically detecting code smells with machine learning techniques.

The proposed framework can mitigate two types of validity threats in software quality experiments: (i) internal validity threats due to instrumentation, since the same measurement instruments can be used in replications, thus not affecting the validity of the results, and (ii) external validity threats due to reduced generalizability, since different researchers can more easily replicate experiments with different settings, populations, and contexts while reusing the same scientific workflow.

Keywords: scientific workflow · software quality · quality assessment · replication · code smells · open source

1 Introduction

1.1 Replication

In Science, replication refers to *"a conscious and systematic repeat of an original study"* [16]. Replication is an important process in Software Engineering research, and many authors have emphasized its relevance [14]. For instance, Kitchenham [15] claims that *"replication is a basic component of the scientific method, so it hardly needs to be justified."*. For Shull et al. [24], replication allows to *"better understand Software Engineering phenomena and how to improve the practice of software development. One important benefit of replications is that they help mature Software Engineering knowledge by addressing internal and external validity problems"*. The same authors also mention that in terms of external validation, replications help to generalize the results, demonstrating that they do not depend on the specific conditions of the original study. Regarding internal validity, replications also help researchers show the range of conditions under which experimental results hold.

However, replication is not consensual. Some authors like Shepperd [23] argue that *"replication is often used to gain confidence in empirical findings, as opposed to reproduction where the goal is showing the correctness, or validity of the published results."* Thus, almost all replications are confirmatory because the prediction intervals are wide. Shepperd suggests to *"limit replications to matters of reproducibility (where warranted)"* [23].

One of the framework's objectives presented in this paper is to enhance the replication/reproduction of studies repeating a previously performed experiment.

1.2 Software Quality Assessment

In software development and maintenance, especially in complex systems, the existence of code smells jeopardizes the quality of the software and hinders several operations, such as the reuse of code.

Code smells are not bugs since they do not prevent a program from functioning, but rather symptoms of software maintainability problems [30]. They often correspond to the violation of fundamental design principles and negatively impact its quality.

Software development and maintenance is a complex task that can be hindered by the presence of code smells [26,30], causing code misunderstanding, therefore reducing maintenance efficiency and promoting defects injection. Their removal can be achieved through refactoring operations, thereby improving software quality, such as reusability, ease of maintenance, and readability [9].

Code smells detection is not an easy task because it requires a lot of effort if the process is entirely manual [17]. Depending mainly on the size and complexity of the source code and the developer's experience, the greater the experience of the latter, the easier it is to detect code smells, as well as the greater the complexity of the detected code. [18,19].

Although there has been some progress in recent years in the detection and visualization of code smells [22], the main problems remain the same: 1) the subjectivity of the detection process, which makes it very manual, 2) difficulties in process automation, with lacking in data for models calibration, 3) absence of detection and visualization tools in the IDE to help developers.

In recent years we have seen an evolution in the automatic detection of smells, with various automation techniques based on machine learning, to be applied

[2,8,9,17,20,21,26], but remains its detection difficult because of two main problems: 1) there is no formal definition of code smell, according to Wang, "Automatic detection of code smells has been studied to help users find which parts of their application codes should be refactored. However, code smells have not been defined in a formal manner" [27]; 2) the calibration of detection algorithms is a key point to good accuracy, and for such the existence of reliable data is needed.

In this context, we focus on software quality, namely the automatic detection of code smells with machine learning.

1.3 Scientific Workflows

Scientific workflows provide a visual representation of the research process, including data inputs, processing steps, and outputs, and enable researchers to organize and automate complex tasks. Enacting scientific workflows can help to improve the reproducibility, efficiency, collaboration, quality control, and reusability of research, leading to more reliable and impactful research results.

Defining scientific workflows has several advantages in research:

- Reproducibility: Scientific workflows define a clear and systematic process for carrying out research, which enables others to reproduce the research results. By making the workflow explicit and transparent, researchers can ensure that others follow the same steps and obtain the same results.
- Efficiency: Scientific workflows provide a structured and efficient way to carry out research. By breaking down the research process into smaller, manageable steps, workflows enable researchers to identify bottlenecks, optimize resource utilization, and reduce the time and effort required to complete the research.
- Collaboration: Scientific workflows facilitate collaboration among researchers by providing a common framework for data and information exchange. By standardizing the data and methods used in the research, workflows enable researchers to share data and result more easily and collaborate on common research objectives.
- Quality Control: Scientific workflows help to ensure the quality and accuracy of research results. By standardizing methods and data collection, workflows can help to minimize errors and inconsistencies and enable researchers to identify and correct errors more easily.
- Reusability: Scientific workflows can be reused for different research projects, saving time and effort in future research. By standardizing methods and data collection, workflows can be adapted for use in different research contexts and as a starting point for future research projects.

The Taverna workflow tool suite was designed to combine distributed Web Services and/or local tools into complex analysis pipelines [28]. Taverna Workflows can be designed and executed on local desktop machines through the *Taverna Workbench*, or they can be executed through other clients or web interfaces using the *Taverna Server*. *Taverna Workbench* connects to *myExperiment*, allowing us to import and export workflows directly from this collaborative environment. *myExperiment* is a Virtual Research Environment for collaboration and sharing workflows and experiments [5].

1.4 Contribution

In this study, we present a framework supported by two open source software, *Taverna Server/Taverna Workbench* [28] and *JGU WEKA REST Service*[1], which allows the construction of workflows for the replication/reproduction of quality studies based on machine learning. To demonstrate the availability of the framework, we present the replication of a study that aims to understand the best machine learning algorithm in code smells detection. It should be noted that other applications can be used to create this framework. For example, we can use *Pegasus* [7] instead of using*Taverna Server* to manage the workflows.

Our goal is to contribute to a solution to the problem presented by Ivie and Thain [13] where they claim, "For a workflow that can run on a single machine, solutions exist for replicability, but for distributed systems, and/or for reproducibility, existing systems are generally inadequate for scientists who are experts in their scientific domain and not experts on computer systems."

The paper is organized through the following Sections: in 2, we introduce some related works; in 3, we describe our framework based on *Taverna and Weka Servers*; in 4, we present an example of replication based on our framework. Finally, Sect. 5 provides some conclusions and future work.

2 Background

Although replication/reproduction of studies is widely considered an essential requirement of the scientific process and plays an important role in building knowledge in Software Engineering, several serious concerns have recently been raised. Thus, in Software Engineering, the use of workflows for the replication/reproduction of studies is still limited and far from standard practice.

Unlike in other areas, e.g., bioinformatics and biomedical sciences, where the use of Scientific workflows management systems are increasingly used [3,29], in Software Engineering is not yet a common practice. For example, searching at *myExperiment*, workflows for Software Engineering, we found very few in the universe of nearly 4.000 workflows in this collaborative environment.

Ivie and Thain [13] describe a set of problems that make replication difficult, such as technical barriers. These authors claim that, in principle, it should be possible to specify a computation to sufficient detail that anyone should be able to reproduce the study exactly. But in practice, there are fundamental, technical, and social barriers to doing so. They also present how various authors have defined reproducibility and the need for more consensus on this subject. Another significant contribution of this study is the chapter "technical barriers to reproducing a workflow", which presents the advantages of using scientific workflows and presents a set of recommendations for their use.

De Magalhães et al. [4] performed a systematic mapping study where they analyzed 37 papers reporting studies about replication published in the last 17

[1] This application is developed by the Institute of Computer Science at the Johannes Gutenberg University Mainz, inserted in the *openrisk* project.

years. The purpose of this paper is to understand the current state of work on replication in empirical Software Engineering research. These authors have concluded that for replication to be a frequently used means of achieving robust results, it still has a long way to go. In replication, there is not yet a set of standardized concepts and terminology, so it is important that the Software Engineering research community engage in an effort to create and evaluate taxonomy, frameworks, guidelines, and methodologies to support replications' development fully.

Regarding the replication of studies, there are other important works on this subject, which emphasize the importance of replication, defined concepts, and guidelines for performing replication [1,11,12,14,23].

Regarding the use of scientific workflows, several works point out the challenges and recommendations for their use [6,25,29].

There are some applications that allow machine learning workflows to be created for users who are not experts on computer systems. Still, these applications are not free or are not for server system architectures. Some of these systems will be presented below.

Azure Machine Learning Studio[2] is a web-based integrated development environment (IDE) for developing data experiments. It is a collaborative, drag-and-drop tool to build, test, and deploy predictive analytics solutions on your data. The user can develop a predictive analytics model by creating a machine learning workflow only by drag-and-drop blocks.

Weka[3] is, according to the authors, "a collection of machine learning algorithms for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization.". Weka is a local open source software.

RapidMiner[4] presents a concept similar to Azure Machine Learning Studio for creating machine learning workflows. Provides products for local operation, such as RapidMiner Studio (visual workflow designer for predictive models), and for servers with RapidMiner Server (Share and re-use predictive models, automate processes, and deploy models into production). RapidMiner is not open source software.

Orange[5] is an open source machine learning and data visualization that uses the same concept as Azure and RapidMinder to create machine learning processes. Orange only works locally.

H20.ai[6] offers a full suite of products designed to make it easy for every user and every enterprise to accelerate the adoption of Machine Learning and artificial intelligence, providing a web-based platform.

None of the above solutions cumulatively has the flexibility of our framework in terms of 1) costs, as we only use free software; 2) possibilities of use, since

[2] https://studio.azureml.net/.

[3] https://www.cs.waikato.ac.nz/ml/index.html.

[4] https://rapidminer.com/.

[5] http://orange.biolab.si/.

[6] https://www.h2o.ai/.

we can build our interface if we do not intend to use *Taverna Workbench*; 3) It is not a local solution, and can be all installed in microservices; 4) ease of availability of scientific workflows in *myExperiment*.

3 Description of the Proposed Framework

3.1 Introduction

The model we present is based on creating workflows of the experiences performed in the studies and making these workflows publicly available. In this way, we allow the entire community to replicate the experiences and confirm the results obtained.

The proposed framework is based on the use of *Taverna* for creating scientific workflows and *JGU WEKA REST Service* for creating machine learning models.

To create a workflow, we first use *Taverna Workbench* (Fig. 1) to design the workflows and perform the first tests, ensuring that the workflow works. Our workflow is essentially a task set that interacts with *Weka Server* through Web Services, in this case, RESTful Web Services.

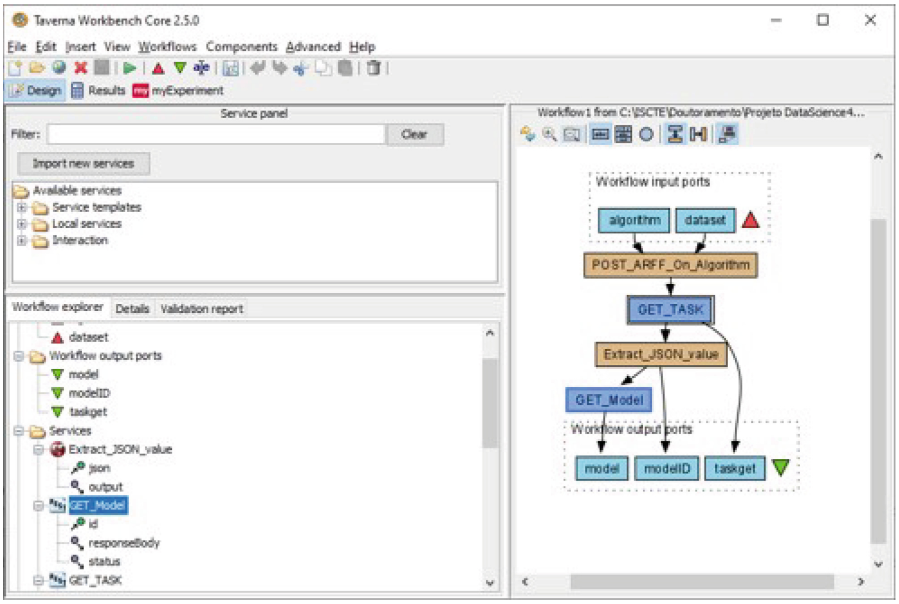

Fig. 1. Taverna Workbench.

Figure 2 shows an example of a workflow that produces the training of a Weka machine learning algorithm by creating its model. This workflow has as inputs the algorithm to be used (e.g.,"J48", "RandomForest", "libsvm", etc.) and the training dataset. The outputs are the model, model ID, and task ID. Creating workflows in *Taverna Workbench* consists of choosing the components you want to add to the workflow from the menus since the interface is graphical.

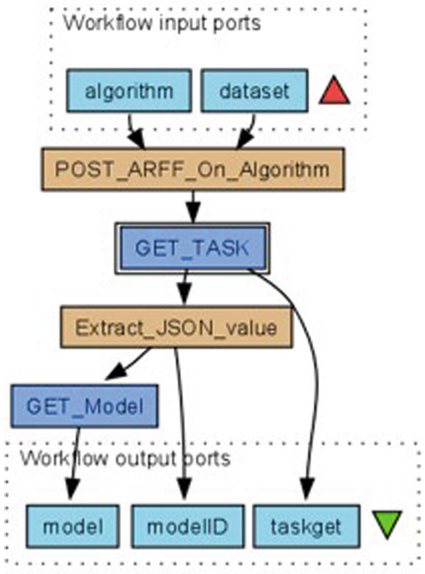

Fig. 2. Workflow to create the Machine Learning model.

Taverna Workbench. Works in local mode, i.e., on local desktop machines, so using very large datasets can cause the machine to slow down or even not have enough resources for Workflow processing. In the second phase, we use *Taverna Server* to solve this problem. The process is to create workflows in *Taverna Workbench* and then import and run them in *Taverna Server*. As the *Taverna Server* is in the cloud and the *Weka Server*, we have no problems with processing speed or lack of resources.

With this architecture, we are convinced that we contribute to solving the problem presented by Ivie and Thain [13] when they say, "For a workflow that can run on a single machine, solutions exist for replicability, but for distributed systems, and/or for reproducibility, existing systems are generally inadequate for scientists who are experts in their scientific domain and not experts on computer systems.".

3.2 Architecture of the Framework

The framework is based on two docker containers, one for *JGU WEKA REST Service* and another for *Taverna Server*, communicating through RESTful API (Application Program Interface), as illustrated in Fig. 3.

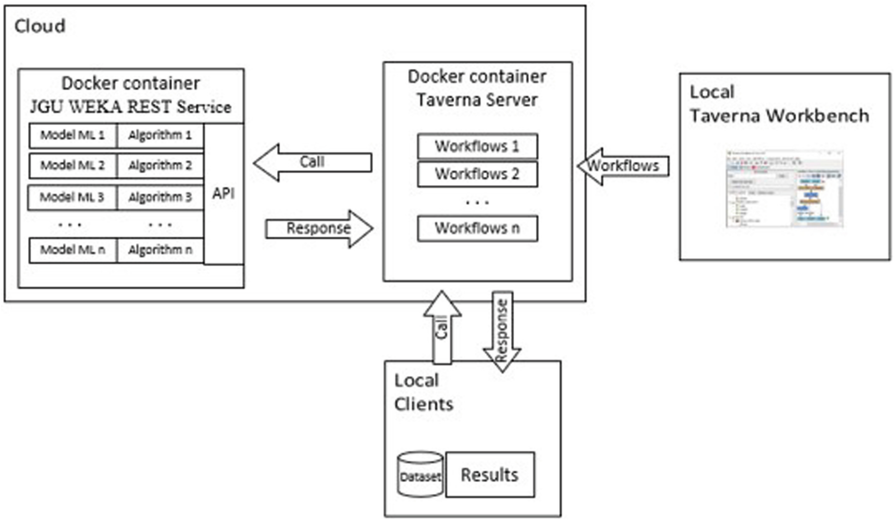

Fig. 3. Framework architecture.

As already mentioned, *Taverna Workbench* works in local mode and serves to produce the workflows that are then loaded into the *Taverna Server*. Once the models are in *Taverna Server*, they are available to be called by different local clients. When a client calls a Workflow in the *Taverna Server*, it interacts with *JGU WEKA REST Service*, which is in another docker container, through the API provided by *Weka Server*.

The workflows we created are for software quality assessment replication using Machine Learning techniques (see Fig. 2), although the same principle holds true for other experiments. Thus, the operating principles are as follows:

a) The workflow that is in the *Taverna Server* receives the ML algorithm and the training dataset as input. These parameters are stored in the *Taverna Server*;
b) Invoking the *JGU WEKA REST Service* API, we pass Weka the two parameters from the previous point, and Weka performs the training of the algorithm with the dataset;
c) *JGU WEKA REST Service* returns the result in a JSON format containing various information, such as training success, model ID, etc.;
d) In *Taverna Server* we will extract the various information from JSON;
e) Obtain from *JGU WEKA REST Service* the created ML model and its evaluation, passing as a parameter the model ID;

This framework presents a scalable solution where it is possible to add more docker containers with other services, thus creating the possibility to produce more elaborate workflows with more functionalities.

We are currently developing a Java graphical user interface so that users can easily interact with the *Taverna Server*. Currently, the workflows placement in the *Taverna Server* is performed manually, and the results are consulted in the server directories, also manually, through a browser. However, it should be noted that it is always possible to invoke a Workflow directly from the *Taverna Workbench* (see Fig. 1), particularly when datasets are small and do not require large computational resources.

4 Replication Example

To exemplify our framework, we have replicated an article that compares various machine learning algorithms to determine the best algorithm for detecting code smells.

4.1 Study to Be Replicated

We chose to replicate Fontana's paper *"Code Smell Detection: Towards a Machine Learning-based Approach"* [10]. This paper has conducted a study where the authors use six different machine learning algorithms to detect four code smell types, i.e., Data Class, Large Class, Feature Envy, and Long Method.

This study aims to compare and benchmark code smell detection with supervised machine learning techniques. For this purpose, four datasets were created, one for each code smell, to train the machine learning algorithms. The application used in this experiment to train and evaluate machine learning algorithms was Weka (open source software from Waikato University), and the following algorithms available in Weka were implemented:

- J48 is an implementation of the C4.5 decision tree, and its three types of pruning techniques: pruned, unpruned, and reduced error pruning;
- JRip implements a propositional rule learner;
- Random Forest consists of a large number of individual decision trees that operate as an ensemble;
- Naïve Bayes is a probabilistic model based on the Bayes theorem;
- SMO is a Sequential Minimal Optimization algorithm widely used for training support vector machines;
- LibSVM is a library for Support Vector Machines (SVMs), integrated software for support vector classification.

Experiments were performed to evaluate the performance values of the machine learning algorithms with their default parameters for each type of code smell. Cross-validation was the statistical technique applied to test the performance of Machine Learning models, and three standard performance measures were used: accuracy, F-Measure, and Area Under ROC (Receiver Operating Characteristics).

The results show that the experienced algorithms obtained high performances, regardless of the type of code smell. On the other hand, the SVM algorithms tend to perform worse than the other algorithms, with the J48 and Random Forest classifiers obtaining the highest accuracy (>96 %).

4.2 Replication Using the Framework

With our framework, it is easy and accessible for everyone to replicate this type of study because the use of scientific workflows avoids repetitive work and having to understand the application used in the training and evaluation of different machine learning algorithms, Weka, in this case.

The replication facility is essentially in the fact that we use the workflow shown in Fig. 2, having as input parameters the desired algorithm and the training dataset and as output parameters the model with its performance measures. Thus, to obtain the results for the different algorithms, just change the two input parameters. For example, to test the performance of the J48 algorithm in detecting the code smell Data Class, it is only necessary to invoke the workflow, passing as inputs the algorithm identifier - in this case, "J48" - and the training dataset previously prepared for this code smell in ARFF format. An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software[7].

Some of the possible values for the first workflow parameter, i.e., the algorithm identification, are RandomForest, J48, J48/adaboost, NaiveBayes, NaiveBayes/adaboost, NaiveBayes/bagging, libsvm, libsvm/adaboost, libsvm/bagging, SMO.

Note that the JRip algorithm is not implemented in our version of *JGU WEKA REST Service*, so it is impossible to compare this algorithm's results.

Regarding the second workflow parameter, i.e., the dataset, the four datasets provided by the paper authors were used, one for each code smell. The dataset is at the class level for Data Class and Large Class code smells, and for Feature Envy and Long Method code smells, the datasets are at the method level.

Each dataset contains the identification of the Java project used, the package, class, and method (the method for code smell only at this level), the code metrics - of class or method - and the value of whether or not it is a code smell.

The results presented in Fontana's paper [10] show that J48, Random Forest, JRip, and SMO have accuracy values greater than 90% for all datasets, and on average, they have the best performances. Naive Bayes has slightly lower performances on Data Class and Feature Envy than on the other two smells. LibSVM performances are lower than the others (far lower in three cases out of four).

[7] https://www.cs.waikato.ac.nz/ml/weka/arff.html.

Table 1. Performance results for Data Class code smell.

Classifier	Accuracy	F-measure	ROC Area
J48	0.981	0.981	0.975
Random Forest	0.974	0.974	0.999
Naive Bayes	0.786	0.792	0.935
SMO	0.952	0.952	0.945
LibSVM	0.738	0.677	0.609

Table 2. Performance results for God Class code smell.

Classifier	Accuracy	F-measure	ROC Area
J48	0.969	0.969	0.961
Random Forest	0.979	0.979	0.990
Naive Bayes	0.931	0.932	0.976
SMO	0.955	0.954	0.943
LibSVM	0.667	0.533	0.500

Table 3. Performance results for Feature Envy code smell.

Classifier	Accuracy	F-measure	ROC Area
J48	0.938	0.938	0.943
Random Forest	0.919	0.919	0.983
Naive Bayes	0.850	0.850	0.912
SMO	0.900	0.898	0.870
LibSVM	0.676	0.566	0.520

Tables 1, 2, 3, 4 present our results for the different machine learning algorithms for each of the 4 code smells. We do not configure any of its parameters in these algorithms, having used the default settings. To evaluate the performance of each model, we used 10-fold cross-validation as the study authors.

We can see slight differences in values when comparing our results with those of the original study. But when we compare the original study's findings, we can confirm them.

The first result obtained by the authors of the original study was that the J48, Random Forest, and SMO algorithms have accuracy values greater than 90% for all datasets, and on average, they have the best performances. We confirm this first result. The second result is that the Naive Bayes algorithm has slightly lower performances for all four code smells. We also confirm this result. The final result is that LibSVM's performance is lower than the other algorithms (much lower in three cases out of four). As we can see in Tables 1, 2, 3 and 4, the worst performing algorithm is LibSVM, and for God Class, Feature Envy

Table 4. Performance results for Long Method code smell.

Classifier	Accuracy	F-measure	ROC Area
J48	0.995	0.995	0.999
Random Forest	0.993	0.993	1.000
Naive Bayes	0.936	0.937	0.964
SMO	0.976	0.976	0.971
LibSVM	0.686	0.576	0.529

and Long Method code smell, results are much lower than the other algorithms. Thus, we also confirm this last result.

5 Conclusions

This study presents a framework that allows the replication/reproduction of studies based on scientific workflows.

Unlike in other areas, e.g., bioinformatics and biomedical sciences, where the use of Scientific workflows management systems has been long used [3,29], in Software Engineering is not yet a common practice. With this framework, we aim to present a simple and scalable way to create and reuse scientific workflows.

This framework is based on two open source software systems: *Taverna Server/Taverna Workbench* [28] and *JGU WEKA REST Service*. This architecture allows workflows to be performed on the local computer that does not require large computational resources. Remote execution on the cloud-based architecture allows the processing of large datasets. Workflows are created in the *Taverna Workbench*, where they can also be run, and loaded into the *Taverna Server*, so they are available to be run by multiple users.

To exemplify the feasibility of the proposed framework, we have replicated the study named *"Code Smell Detection: Towards a Machine Learning-based Approach"* published in [10], which compares six machine learning algorithms to determine the best one for detecting 4 code smell types, i.e., *Data Class, Large Class, Feature Envy* and *Long Method*. Comparing our results with the study's, we confirm its findings.

With this framework, we think we have contributed to facilitating the application/reproduction of studies, thus contributing to Software Engineering development.

Acknowledgement. This research was partially funded by ISTAR's projects FCT UIDB/04466/2020 and UIDP/04466/2020.

References

1. Abbuhl, R.: Why, when, and how to replicate research. In: Research Methods in Second Language Acquisition: A Practical Guide, pp. 296–312 (2012). https://doi.org/10.1002/9781444347340.ch15

2. Bryton, S., Brito e Abreu, F., Monteiro, M.: Reducing subjectivity in code smells detection: experimenting with the Long Method. In: Proceedings of the 7th International Conference on the Quality of Information and Communications Technology (QUATIC), pp. 337–342. IEEE (2010). https://doi.org/10.1109/QUATIC.2010.60
3. Cohen-Boulakia, S., Chen, J., Missier, P., Goble, C., Williams, A.R., Froidevaux, C.: Distilling structure in taverna scientific workflows: a refactoring approach. BMC Bioinformatics **15**(Suppl 1), 1–14 (2014). https://doi.org/10.1186/1471-2105-15-S1-S12
4. De Magalhães, C.V., Da Silva, F.Q., Santos, R.E., Suassuna, M.: Investigations about replication of empirical studies in software engineering: a systematic mapping study. Inf. Softw. Technol. **64**, 76–101 (2015). https://doi.org/10.1016/j.infsof.2015.02.001
5. De Roure, D., Goble, C., Stevens, R.: The design and realisation of the Experimentmy virtual research environment for social sharing of workflows. Futur. Gener. Comput. Syst. **25**(5), 561–567 (2009). https://doi.org/10.1016/j.future.2008.06.010
6. Deelman, E., et al.: The future of scientific workflows. Int. J. High-Perform. Comput. Appl. **32**(1), 159–175 (2018). https://doi.org/10.1177/1094342017704893
7. Deelman, E., et al.: Pegasus, a workflow management system for science automation. Future Gener. Comput. Syst. **46**, 17–35 (2015). https://doi.org/10.1016/J.FUTURE.2014.10.008
8. Fokaefs, M., Tsantalis, N., Stroulia, E.: JDeodorant: identification and application of extract class refactorings. In: Proceedings of the 33rd International Conference on Software Engineering, (ICSE). ACM/IEEE (2011). https://doi.org/10.1145/1985793.1985989
9. Fontana, F.A., Mangiacavalli, M., Pochiero, D., Zanoni, M.: On experimenting refactoring tools to remove code smells. In: Proceedings of the XP'15 Workshops, pp. 1–8. ACM Press, New York (2015). https://doi.org/10.1145/2764979.2764986
10. Fontana, F.A., Zanoni, M., Marino, A., Mäntylä, M.V.: Code smell detection: towards a machine learning-based approach. In: Proceedings of the International Conference on Software Maintenance (ICSM). IEEE (2013). https://doi.org/10.1109/ICSM.2013.56
11. Gómez, O.S., Juristo, N., Vegas, S.: Understanding replication of experiments in software engineering: a classification. Inf. Softw. Technol. **56**(8), 1033–1048 (2014). https://doi.org/10.1016/j.infsof.2014.04.004
12. Harman, M., McMinn, P., de Souza, J.T., Yoo, S.: Search based software engineering: techniques, taxonomy, tutorial. In: Meyer, B., Nordio, M. (eds.) LASER 2008-2010. LNCS, vol. 7007, pp. 1–59. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-25231-0_1
13. Ivie, P., Thain, D.: Reproducibility in scientific computing. ACM Comput. Surv. **51**(3), 1–36 (2018). https://doi.org/10.1145/3186266
14. Juristo, N., Gómez, O.S.: Replication of software engineering experiments. In: Meyer, B., Nordio, M. (eds.) Empirical Software Engineering and Verification. Lecture Notes in Computer Science, vol. 7007, pp. 60–88. Springer, Berlin (2012)
15. Kitchenham, B.: The role of replications in empirical software engineering-a word of warning. Empir. Softw. Eng. **13**(2), 219–221 (2008). https://doi.org/10.1007/s10664-008-9061-0
16. La Sorte, M.A.: Replication as a verification technique in survey research: a paradigm. Sociol. Q. **13**(2), 218–227 (1972). https://doi.org/10.1111/j.1533-8525.1972.tb00805.x

17. Liu, H., Ma, Z., Shao, W., Niu, Z.: Schedule of bad smell detection and resolution: a new way to save effort. IEEE Trans. Softw. Eng. **38**(1), 220–235 (2012). https://doi.org/10.1109/TSE.2011.9
18. Mantyla, M., Lassenius, C.: Subjective evaluation of software evolvability using code smells: an empirical study. Empir. Softw. Eng. **11**(3), 395–431 (2006). https://doi.org/10.1007/s10664-006-9002-8
19. Mantyla, M., Vanhanen, J., Lassenius, C.: Bad smells - humans as code critics. In: Proceedings of the 20th International Conference on Software Maintenance (ICSM), pp. 399–408 (2004). https://doi.org/10.1109/ICSM.2004.1357825
20. Palomba, F., Bavota, G., Penta, M.D., Oliveto, R., Poshyvanyk, D., Lucia, A.D.: Mining version histories for detecting code smells. IEEE Trans. Software Eng. **41**(5), 462–489 (2015). https://doi.org/10.1109/TSE.2014.2372760
21. Pessoa, T., Brito e Abreu, F., Monteiro, M.P., Bryton, S.: An eclipse plugin to support code smells detection. In: Proceedings of INFORUM 2011 (Simpósio de Informática). p. 12 (2011). https://arxiv.org/abs/1204.6492
22. Pereira dos Reis, J., Brito e Abreu, F., de Figueiredo Carneiro, G., Anslow, C.: Code smells detection and visualization: a systematic literature review. Arch. Comput. Methods Eng. **29**(1), 47–94 (2022). https://doi.org/10.1007/s11831-021-09566-x
23. Shepperd, M.: Replication studies considered harmful. In: Proceedings of the International Conference on Software Engineering (ICSE), pp. 73–76. ACM/IEEE (2018). https://doi.org/10.1145/3183399.3183423
24. Shull, F.J., Carver, J.C., Vegas, S., Juristo, N.: The role of replications in empirical software engineering. Empir. Softw. Eng. **13**(2), 211–218 (2008). https://doi.org/10.1007/s10664-008-9060-1
25. Taylor, I.J., Deelman, E., Gannon, D., Shields, M.S.: Workflows for E-science: Scientific Workflows for Grids. Springer, Cham (2007). https://doi.org/10.1007/978-1-84628-757-2
26. Tsantalis, N., Chaikalis, T., Chatzigeorgiou, A.: JDeodorant: identification and removal of type-checking bad smells. In: Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR), pp. 329–331 (2008). https://doi.org/10.1109/CSMR.2008.4493342
27. Wang, C., Hirasawa, S., Takizawa, H., Kobayashi, H.: Identification and elimination of platform-specific code smells in high performance computing applications. Int. J. Networking Comput. **5**(1), 180–199 (2015). https://doi.org/10.15803/ijnc.5.1_180
28. Wolstencroft, K., et al.: The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Res. **41**(Web Server issue), 557–561 (2013). https://doi.org/10.1093/nar/gkt328
29. Wolstencroft, K., Fisher, P., Goble, C.: Scientific workflows overview. Connexions **26**, 1–6 (2009)
30. Yamashita, A., Moonen, L.: To what extent can maintenance problems be predicted by code smell detection? - An empirical study. Inf. Softw. Technol. **55**(12), 2223–2242 (2013). https://doi.org/10.1016/j.infsof.2013.08.002