Towards an Extensible Architecture for an Empirical Software Engineering Computational Platform

Fábio Fagundes Silveira[1(✉)], Rodrigo Avancini[1], David de Souza França[1], Eduardo Martins Guerra[2], and Tiago Silva da Silva[1]

[1] Federal University of São Paulo – UNIFESP, São José dos Campos, Brazil
fsilveira@unifesp.br
[2] Free University of Bolzen-Bolzano – UNIBZ, Bozen-Bolzano, Italy

Abstract. One of the main objectives of Empirical Software Engineering is the improvement of methods, techniques, and tools for Software Engineering (SE) through the conduction and execution of experiments. Although research in empirical software engineering is well established in the literature, it is possible to verify that there are few supporting environments for conducting experiments, despite the increasing demand for this kind of study within the SE community. Moreover, existing approaches lack a number of important features, such as extensible mechanisms for the required phases of an experimentation in different domains of SE, integration with existing software development environments, ways to define metrics to be collected, automatic organization mechanisms to analyze collected data using a specific statistical analyzes method, and remote execution of experiments. This paper describes the *SCCOPLA* – an *extensible architecture for empirical software engineering computational platform*. Our approach aims to address the aforementioned functionalities missing in available environments to conduct empirical SE studies. Preliminary results show its feasibility by applying an architecture-based tool to some of our previous software testing empirical studies.

Keywords: Empirical software engineering · Controlled experiments · Experimental support environment

1 Introduction

Improving methods, techniques, and tools for Software Engineering (SE) through the conduction and execution of experiments are some of the main objectives of Empirical Software Engineering. Although research in empirical software engineering is well established in the literature, it is remarkable that there are few supporting environments for conducting experiments, despite the increasing

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12957, pp. 231–246, 2021.
https://doi.org/10.1007/978-3-030-87013-3_18

demand for this kind of study within the SE community [8,10,30]. Additionally, existing approaches lack a number of important features. For instance, they do not offer extensible mechanisms for all required phases of experimentation in the different domains of SE, integration with existing software development environments, ways of how to define metrics to be collected, automatic organization mechanisms to analyze collected data using a specific statistical analyzes method, and remote execution of experiments. Notably, the capability to carry out remote experiments has become much more crucial after the onset of the COVID-19 pandemic and subsequent lockdowns.

It is essential to highlight that experiments have specific characteristics regarding their domains. These differences affect how computational environments that support empirical studies need to be extensible, allowing the highest possible flexibility to be tailored to different domains.

In this paper, we describe the *SCCOPLA* – an *exten**s**ible ar**c**hitecture for empiri**c**al s**o**ftware engineering com**p**utational p**la**tform*. Our approach aims to address the aforementioned functionalities missing in available environments to conduct empirical SE studies. Although our proposed architecture is an ongoing research and development project, preliminary results show its feasibility by applying an architecture-based tool to some of our previous software testing empirical studies.

The remaining of this paper is structured as follows: Sect. 2 reports the background notions related to empirical software engineering. Section 3 describes the extensible architecture for conducting SE experiments proposed in this work. Section 4 describes the tool developed to implement *SCCOPLA* and its main components. Also, discusses the present status, current results of our research and points out the next steps of the project. Finally, Sect. 5 summarizes our conclusion.

2 Empirical Software Engineering

A typical process of a controlled empirical study comprises the following phases: definition, planning, operation, analysis, interpretation, and packaging [29]. The seminal paper highlighting the importance of empirical studies in software engineering was published in the '80s by Basili et al. [3]. After that, several studies have been conducted in this area and reported in the literature.

Lately, the use of proof of concepts might not be enough in the assessment of proposed ideas in software engineering. This way, over the past years, the SE community has systematically increased its need for applying rigorous and systematic empirical studies, aiming to assess new scientific evidence about methods, techniques, and software technology proposals [21]. Reducing the risks involved in technology transfer to the industry is one of the clear benefits of such an approach. This way, hundreds of experiments have been conducted in the SE area [24].

Several researches on ESE applications show up important works in these areas. However, studies point out the lacking of automated and integrated tools

or computational environments for supporting the experiment process phases [7,8,21,30], despite the growing demand for empirical studies in SE. Arisholm et al. [1] e Karahasanoviæ et al. [14] have proposed the SESE (*Simula Experiment Support Environment*), a web tool to support the participants' management, measures the time spent for each task, enable to download of additional information, and monitor participants activities. Nevertheless, it does not offer support to data acquiring and analyses.

The *Value-Based Empirical Research Planning Framework* (VBER) [4] was developed to help the study planning and evaluation, linking some deliverables to stakeholders propositions, aiming to compare the benefits and risks of empirical study variants. The *Mechanical Turk* [26] comprises a crowdsourcing tool to support empirical studies in SE, offering resources to access and manage a large number of participants, assisting in subjects recruitment to assess a technique or an ES tool. Torii et al. [27] have proposed the Ginger2, an experimental environment based on the CAESE (*Computer-Aided Empirical Software Engineering*) framework. According to the authors, even though the CAESE supports the whole experiment process, Ginger2 is still restricted to the analyses and execution phases. Mendonça et al. [19] have developed the FIRE (*Framework for Improving the Replication of Experiments*), focusing on sharing the knowledge generated by the study in a common body (intra and inter groups). However, they do not suggest how to organize lab packages.

The *Experiment Manager Framework* (EMF), proposed by Hochstein et al. [11] is a set of integrated tools to support experiments for the high-performance computing domain. The experiment framework supports many of the activities involved in a controlled experiment. Anyway, it does have some drawbacks. For example, it is not easy to adapt it to a particular domain language engineering since automated data capturing is customized for high-performance source code only. Another limitation is the lack of support for expanding statistical analyses beyond the ones built in the platform. Costa Araújo et al. [6] have presented the *Application of Reproducible Research on Evaluation of Software Testing Technique* (ARRESTT). Their main goal is to provide and encourage reproducible experimental artifacts, enabling the experimenter to develop, distribute and rerun studies within the software testing domain, such as test case selection and test suite minimization techniques. Although ARREST has several features that help the experiments with some replication issues, their uses have to know java to manipulate framework elements adequately. Neto and Scatalon [20] have shown up the ExpTool, a platform to carry out and packing experimental data in SE. On its current version, data are stored in XML files based on an ontology proposed for empirical studies in the literature [12].

It is possible to find out approaches specifically based on DSLs (*Domain Specific Languages*). For example, Freire et al. [8] have presented the ExpDSL, a tool, and a DSL to support experiments in SE. Basically, the approach encompasses the definition (scope) and the design experiment phases. Häser et al. [10] has developed an integrated end-to-end environment to perform experiments in DSL engineering, focusing on goal definition and experiment planning. As

stated by the authors, the language offers to the experimenter to represent all the relevant data, while the tool automates the generation of interesting plots and testing the hypothesis.

Literature also reports approaches based on ontologies. The eSEE (*Experimentation Environment to Support Large-Scale Experimentation*), proposed by Travassos et al. [28] is an environment to manage several kinds of empirical studies in SE, including knowledge acquired when defining, planning, executing, and packaging studies. The eSSE is based on web services that facilitate instantiating environments to enable certain experimental study activities. It works in three different abstract levels: meta, configured, and execution, presenting an initial set of tools to populate its infrastructure. Garcia et al. [9] have developed an ontology to specify controlled experiments called *experOntology*. The main idea is to formalize the experiment plan through a body of knowledge previously recorded. It can model the design of an experiment, assigning, manually, different treatments to the participants. However, it is presented no empirical study to evaluate the ontology. The TESE (*Tool for Experimental Software Engineering*) [25] comprises a tool that, according to the authors, the main difference is the provision of aid its users about the main concepts involving empirical software engineering. TESE is implemented as a standalone application, which makes it difficult to manage geographically distributed experiments.

3 The *SCCOPLA* (An extenSible arChitecture for empiriCal sOftware Engineering comPutational pLAtform)

This section describes the architecture proposed in this work, particularly its components, their foundations, and how these components are integrated and extended.

During the design of our approach, aiming to fulfill the lacking of the flexibility of the existent approaches, the following main requirements were identified (reduced due to space limitations): *R1*) the system should support the whole workflow of a local or distributed controlled experiment, including online monitoring of participants; *R2*) the system should provide extensibility according to the experiment's study needs; *R3*) the system should provide flexible storage mechanisms; *R4*) ... provide extensible mechanisms to define and automatically collect metrics, preferable non-invasive ones; *R5*) ... provide extensible support to provide statistical analysis design; *R6*) ... provide mechanisms to allow conducting studies, including repeated measures with cross-over experimental design; and *R7*) ... provide extensible capabilities to integrate existing external tools. In particular, this requirement allows the users to either extend the environment to adapt it to different software engineering experiment domains through plugins or integrate it with existing technology solutions.

Considering the aforementioned main requirements of an ESE computational platform, *SCCOPLA* was designed based on microkernel architectural pattern [22]. Besides that, the proposed architecture highly adheres to the rules of the

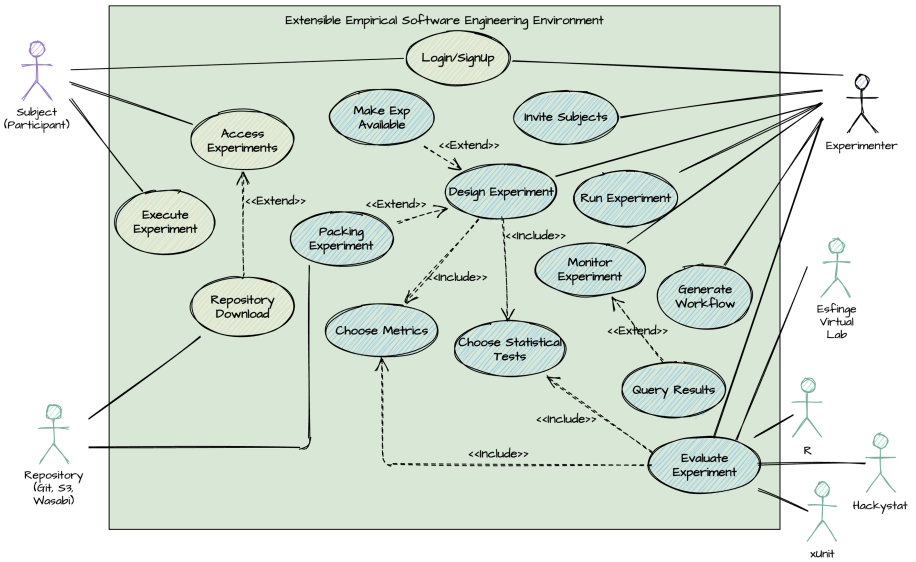


Fig. 1. Use cases diagram of the proposed platform.

Clean Architecture [18] and to the SOLID principles. Functional requirements and boundaries of the *SCCOPLA* can be seen in the use case diagram (Fig. 1).

Primarily, the *SCCOPLA* was sketched to be extended through the addition of plugins. Among the main advantages of this type of architecture, it is worth highlighting the extensibility via well-defined interfaces. Through this mechanism, it is often possible to build plugins without even knowing the source code of the application, being enough to implement the functions (or methods) of the interfaces that the host application is in charge of recognizing and using the plugin when needed.

Figure 2 presents a high-level view of the *SCCOPLA*: the core system and the independent plugin modules integrated into the reference implementation. We have defined custom interfaces (contracts) for each service to be developed adapters to connect to the desired services. This way, the independence of plugins is reached, providing extensibility, flexibility, and isolation to our application core features. This same view (Fig. 2) also presents some concepts of the application and their respective technologies and services available to assist in carrying out scientific experiments. At the right and left ends, one can see users of the application, experimenters, and participants. In the core system, some necessary plugins are displayed, such as: tools for statistical analysis, IDEs for experiments that require execution/development of source code, survey form services, cloud data storage services, and data analysis and manipulation tools.

Aiming to establish the boundaries of the architecture, as suggested by Martin [18], the system was decomposed into components and distributed in layers as depicted in Fig. 3.

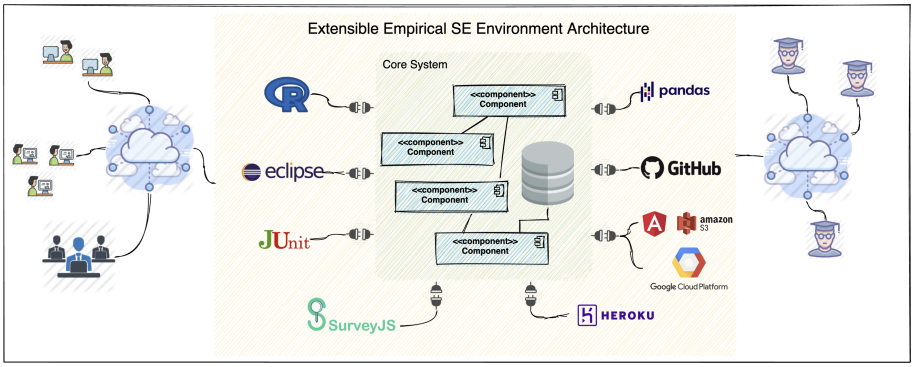


Fig. 2. The *SCCOPLA* architecture model.

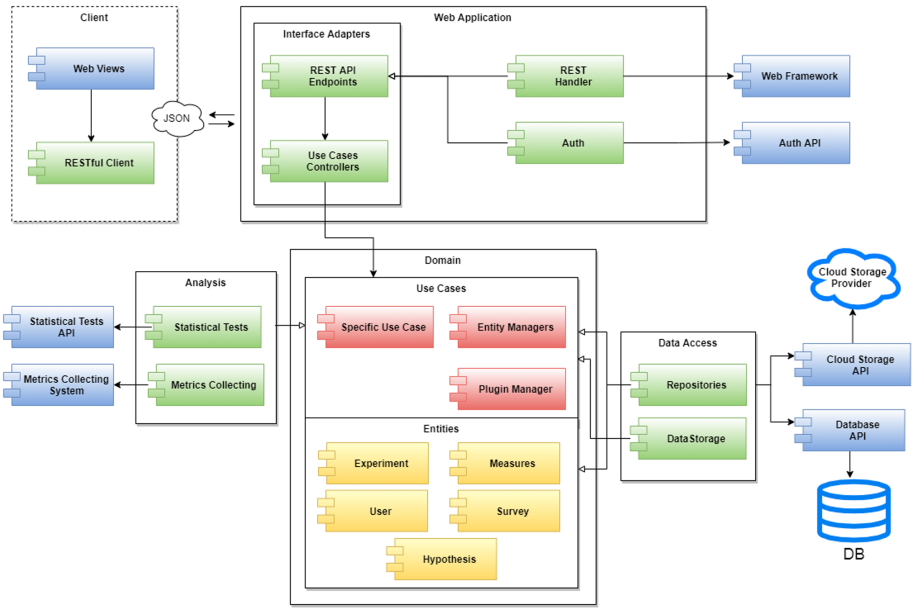


Fig. 3. Components diagram for the proposed platform.

In the Domain layer, entities refer to the business rules of the scientific experiment domain. They encapsulate the most general and high-level application rules. An entity can be an object with methods or a set of data structures and functions. In addition, entities are the most isolated objects and least prone to change. In the Use Cases layer, each use case represents an application-specific business rule. The use cases encapsulate and implement all the application's business logic, orchestrating the data flow using the entities to achieve the user's goals in the application. Here, there are two kinds of use cases: Entity Manager,

responsible for some entity CRUD operations, and Specific Use Case, executed when it is necessary to interact with a more extensive set of entities and other tool resources.

According to the clean architecture guidelines, the Dependency Rule says that when a use case needs to use a resource provided by an API or framework, it uses an interface within its component instead of a concrete object that implements the resource. This rule is used to specify that the dependencies must always point from the lower-level components to the higher-level ones, considering the entities the highest level components.

Although the Clean Architecture ensures that components in the outer layers can act as plugins for business logic, which makes the architecture implicitly pluggable, the use case Plugin Manager has been defined to provide information about plugins used in the tool. Thus, an object that implements an external resource must also inherit from the Plugin interface provided by the Plugin Manager component.

The Web Application layer (or Back-end) provides a RESTful API that conforms to the constraints of REST (Representational State Transfer) architectural style. Thus, this layer is responsible for handling requests and responses for the Client, interacting with the use cases through the controllers. By convention, we define a corresponding endpoint for each entity, which, in turn, has a corresponding controller.

The Back-end also has a mechanism for authentication and authorization of access to endpoints represented by the Auth component. Hence, it prevents the endpoints from being accessed improperly. This mechanism also prevents the use cases from handling all user validation work, which improves the allocation of resources and the overall application's performance.

Regarding the Client layer (or Front-end), we adopted a User-Centered Design approach in which we considered the user's needs as a basis to create and validate user interface possibilities. Thus, in order to meet the end users' goals in accordance with the proposed architecture, the Front-end was developed as a rich client standalone application, totally decoupled from the Back-end, using it only to consume web services provided by the REST API, as discussed in more detail later in this paper.

As shown in Fig. 4, the Clean Architecture is structured like an onion. The inner layers are comprised of more abstract parts, while the outer layers are implementation-specific with the glue code that communicates to the next circle inwards.

Noteworthy that internal elements do not depend on the external ones as it moves to the lower layers. Thus, the level of abstraction increases and the architecture level becomes higher. The most external elements are the concrete elements of the low-level ones (e.g., database and web framework). Figure 5 shows objects involved in the experiment registration.

Decomposing the software into layers and using the dependency rule makes it possible to build an intrinsically testable system with all the benefits it implies. When any of the external parts of the system become obsolete (e.g., database

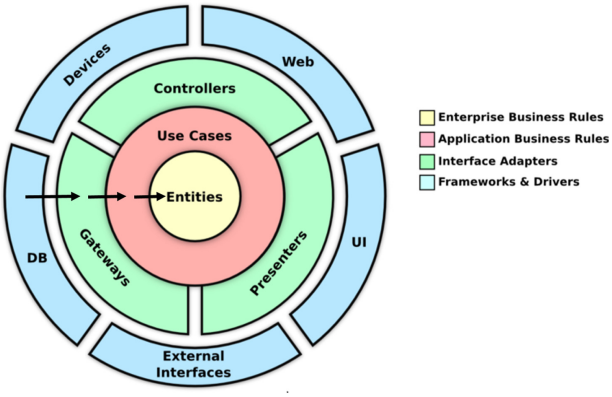


Fig. 4. The clean architecture [18].

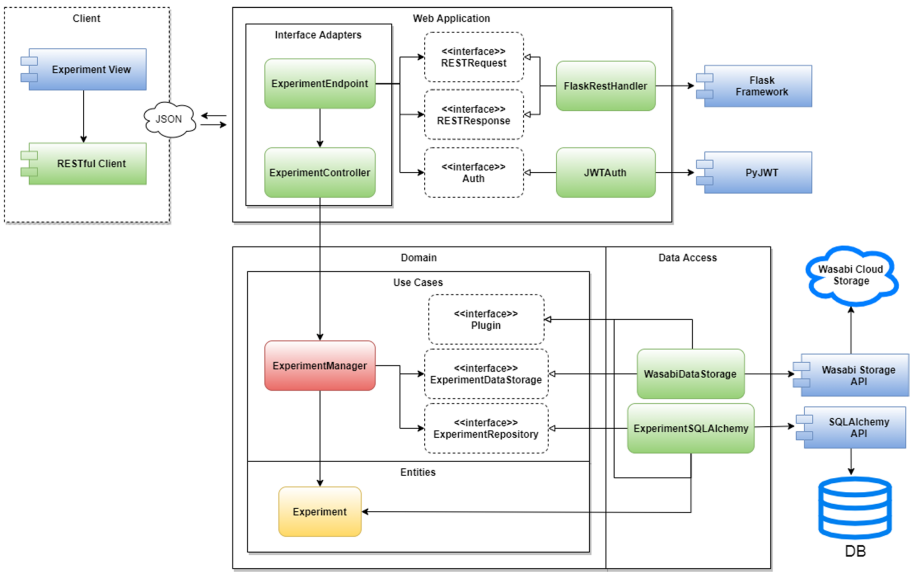


Fig. 5. Source code dependencies always point inwards.

or the web framework), it is possible to replace those obsolete elements with a minimum of maintainability effort.

Throughout the *SCCOPLA* development phases, we aimed to apply concepts from UX (User Experience), DX (Developer Experience), and End-User Development. We adopted a User-Centered Design approach in which we: carried out investigations by interviewing researchers and practitioners to understand their needs; carried out ideation rounds to bring up different ideas, and; prototyped different options to identify challenges and uncover subtleties. In order

to improve the user experience, user testing sessions and questionnaires will be applied when the experiments are carried out.

3.1 Functionalities and Extension Points

One of the main requirements of the infrastructure is the ability to both support experiments with geographically distributed subjects and aggregate inter-institutional research groups. Hence, is mandatory to control and manage all the phases of an experimental study. Aiming to overcome this situation, the architecture implements a monitoring dashboard that presents to the experimenter data about the current status of a study and the status of each enrolled subject, as depicted by Fig. 7. It is essential to notice the UX aspects involved in the front-end GUI design: the accessibility concern. For example, the use of icons instead of only colors to make the GUI color-blind-friendly.

Yet, considering the geographical issues, to enable remote data access both to experimenters as well subjects, we decided to use distributed cloud infrastructure, as Google Cloud Firestore and Amazon S3. Actually, we are using noSQL databases to store the experiments data, since some kinds of experiments deal with non-structured data.

One of the main features of the proposed architecture regards its extensibility. So far, we have designed the following extension points: i) data storage; ii) collecting data forms (e.g., questionnaires); iii) software metrics collecting systems and; iv)statistical tests.

Concerning metrics collecting systems, according to Bykov et al. [5], for now, PRO Metrics (PROM) [23], and Hackystat [13] are the most widely known metric collection systems, developed initially to project monitoring rather than to run experiments. The former comprises a distributed architecture that collects various types of software data: software metrics and personal software process (PSP), an invasive method since PSP requires the participant's involvement. The latter enables automatic collecting software metrics from what authors called "sensors" attached to development tools via SOA. This way, data are sent to the servers to be analyzed. Some metrics include size and defect data. However, this collecting system is only for Java, and it is no longer under active development.

It is necessary to collect both software metrics and participant's behavior data in a most non-invasive way in an experiment monitoring approach. Moreover, since we are developing the system as a web application, it is important not to depend on installing a third-party application on the participant/experiment's computer to collect such metrics. This feature should integrate in a transparent way to the users. Currently, we are investigating how to use the approach proposed by Bykov et al. [5] to attach software sensors in our approach (both on the client and server-side) as observers to the experiment data uploaded to the storage solution plugged into our platform.

Regarding statistical tests extension, we are using an virtual laboratory called *Esfinge Virtual Lab*[1] (EVL). EVL allows users to upload java components to

[1] http://ssrn.inf.unibz.it/virtuallab/.

execute analysis on its plataform, accessing several datasets. This way, statistical tests are implement in java and deployed to the EVL, making them available as web services.

4 *K-Alpha*: A Tool for Managing SE Experiments

In order to provide a proof of concept for the *SCCOPLA*, we developed a tool called *K-Alpha*. This tool consists of a back-end Python Flask-based RESTful web services and a front-end React single-page web application. Figure 6 depicts the deployment diagram for the proposed system.

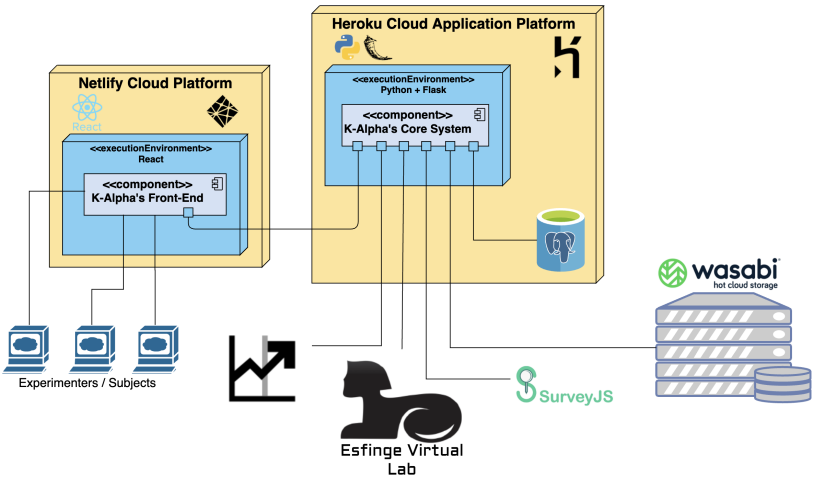


Fig. 6. Deployment diagram for the proposed system.

4.1 The Front-End Application

Among the myriad of possibilities, a single-page web application was chosen to consume the data provided by the endpoints. First of all, the front-end's application has been developed based on three main processes and using the tools provided by the API to do that: the edition of experiments, the application of experiments, and the experiments management. Noteworthy, these three main processes were identified from interviews with SE researchers during a User Research stage.

The edition of experiments includes the creation and edition of experiments to be applied to participants. The experimenter can control issues such as a number of hypotheses, analysis tools used, generation of questionnaires, and adding additional files to guide participants in its executions. Those stages help to create a complete experiment and describe the subject of it to clarify the participants.

The second process is the application experiments, where a set of participants is selected to become the experiment's target. Those participants can access through the web application custom areas to answer questionnaires, count the spent time reading, writing, or whatever the experiment requires, upload and download files needed to perform tasks.

Last but not least, we implemented a dashboard, accessible only to experimenters, where all pieces of information from experiments are compiled to become a valuable tool in managing experiment information. For example, how many and which participants are participating in the experiment, in which state is each of them. The views of the experiment's creation, participant area, and dashboard are shown in Fig. 7.

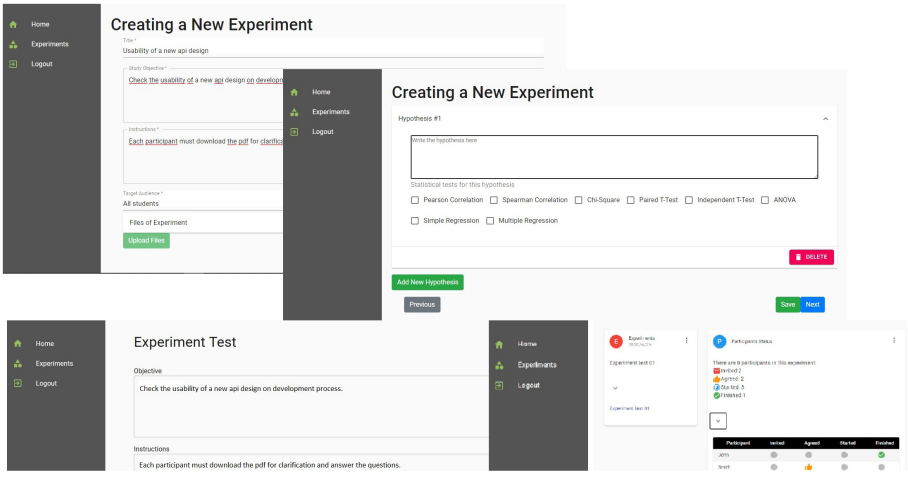


Fig. 7. Creation of a new experiment, addition of hypothesis in the experiment, *K-Alpha* participant's view of the experiment, and *K-Alpha*'s dashboard.

In order to reach the requirements described in Sect. 3, was decided to divide the *K-Alpha* into six components: Rest client: responsible for sending and receiving the messages to the back-end using RESTful services; Authentication: an orthogonal service that intercepts controllers' requests to guarantee the access rules for each profile; Controllers: they hold the business rules of each application screen and control the flow among them; Views: responsible for data presentation and interaction with users (experimenters and participants); Services: work as middlewares between controllers and rest clients. They are responsible for processing the data before sending it to the back-end or views, and; Entities: data models used by the application. Figure 8 shows the diagram with all components mentioned above that compose the front-end application architecture.

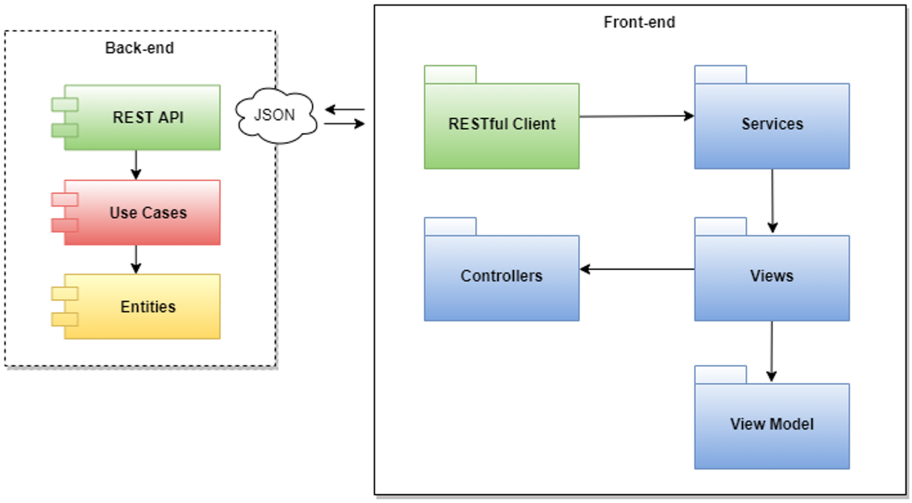


Fig. 8. *K-Alpha* front-end architecture.

4.2 Present Status and Next Steps

As part of the assessment of the approach, we have started reproducing previous experimental studies conducted by our research group [15–17] to observe whether reproductions of experiments are feasible through the proposed computational infrastructure. Preliminary results show that the tool helps guide both the experimenter and the participant to apply the experiment.

Regarding the experimenter, it was much easier to fill the experiment planning, filling out the context, hypotheses, variables, metrics, and experimental design (for now, only cross-over and control group experimental design (or pre-post test) algorithm is implemented. Such facilities are especially significant for those who are learning to use or apply empirical studies.

As our previous experiments are related to software testing, we have implemented an adapter to plug in the *JUnit* framework to the *K-Alpha*, acting as the oracle that implements assertions. This way, it was possible to assess (almost in an automatic way) the source code uploaded by participants to the platform storage system in use (Wasabi). This reduced substantially the effort accomplished by the researcher to manually run participant's test cases.

Regarding the analysis phase, to test the integration of our platform with the EVL, we have deployed Java components to the EVL servers to run the same previously statistic tests: Shapiro-Wilk, Wilcoxon/Mann-Whitney, and paired Student t-test. Then, we have loaded the storage system with the same participant's results to verify if the statistic results differ or not from the expected ones described in our previous work.

Concerning participants, preliminary results indicate a good user experience when guiding the subject through the tasks assigned by the experimenter needed

to conduct the experiment to extract the relevant data. Some of the most frequent questions made by subjects when applying local experiments in our laboratories, regarding, for example, the order of running activities, have practically disappeared. Besides that, automating the data upload to the central storage repository deals with common questions about submitting their data.

Even though some tests with Hackystat have been made (to Java only), we plan to integrate our architecture with the new non-invasive automatic software collecting metrics platform proposed in [5]. The main idea is to keep as much as possible the process of collecting metrics transparent and without requiring the personal involvement of users.

Currently, our efforts are also concentrated in how to implement the experiment packaging phase. Such phase is responsible for storing material, procedures, and data to allow subsequent replication of studies. We plan to implement a "reuse" experiment feature, setting up a new instance of a previous experiment. This way, it will be possible to compare results across different samples of the same study setup. Moreover, enabling users to choose statistic tests without the need for programming skills is on our radar. Here, the key to achieving this feature relies on introducing User-End Development concepts in our platform.

Finally, we are preparing a pilot study to empirically assessing a recent tool developed by our research group to evaluate API usage in the context of software ecosystems [2]. By conducting this study, we aim to assess the extensibility of the *SCCOPLA* for different SE empirical studies domains since evaluating a tool's usefulness differs from assessing testing techniques. The former, for example, need to have tasks set up to ask questions based on the tool's visualization, while the latter involves activities like test cases generation.

5 Conclusion

Nowadays, the development of computational environments to allow geographically distributed SE experiments has become even more critical. Beyond the fact that these platforms could improve the recruitment and participation of subjects in experiments, allowing large-scale studies [1], the COVID-19 pandemic and their physical barriers have enhanced their needs.

Notwithstanding the increased demand for this type of research, the existing few environments lack critical features such as defining metrics to be collected, automatic organization mechanisms to analyze collected data using a specific statistical analysis method, and remote experiment execution.

In this context, we have proposed an extensible architecture for an empirical software engineering computational platform. The *SCCOPLA* aims to address these missing functionalities by using plugins mechanisms. Among the main advantages of this type of architecture, it is worth highlighting the extensibility via well-defined interfaces. These mechanisms enable users to either extend the environment to adapt it to different software engineering experiment domains or integrate it with existing technology solutions.

Preliminary results on applying the proposed approach revealed benefits both to subjects and experimenters, such as guiding subjects through the

tasks assigned by the experimenter and decreasing the effort accomplished by researchers to manually run participants' test cases when reproducing an empirical software testing study.

Future work includes finishing the *K-Alpha* implementation and plugins integration and concludes the replication of our previous experiments using the proposed computational infrastructure. Besides, a new empirical study to assess another recent tool developed by our research group will be conducted using this approach.

Acknowledgment. The authors would like to thank FAPESP (grant 2018/22064-4) for financial support.

References

1. Arisholm, E., Sjøberg, D.I.K., Carelius, G.J., Lindsjørn, Y.: A web-based support environment for software engineering experiments. Nord. J. Comput. **9**(3), 231–247 (2002). http://dl.acm.org/citation.cfm?id=766526.766530
2. Avancini, R., Silveira, F.F., Guerra, E.M., Andrade, P.R.: Software visualization tool for evaluating API usage in the context of software ecosystems: a proof of concept. In: Gervasi, O., et al. (eds.) ICCSA 2020. LNCS, vol. 12254, pp. 335–350. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58817-5_26
3. Basili, V.R., Selby, R.W., Hutchens, D.H.: Experimentation in software engineering. IEEE Trans. Softw. Eng. **12**(7), 733–743 (1986). http://dl.acm.org/citation.cfm?id=9775.9777
4. Biffl, S., Winkler, D.: Value-based empirical research plan evaluation. In: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), pp. 494–494, September 2007. https://doi.org/10.1109/ESEM.2007.50
5. Bykov, A., et al.: Towards non-invasive software measurement system: architecture and implementation. In: Ciancarini, P., Litvinov, S., Messina, A., Sillitti, A., Succi, G. (eds.) SEDA 2016. AISC, vol. 717, pp. 149–165. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-70578-1_15
6. da Costa Araújo, I., da Silva, W.O., de Sousa Nunes, J.B., Neto, F.O.: ARRESTT: a framework to create reproducible experiments to evaluate software testing techniques. In: Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing. SAST, pp. 1:1–1:10. ACM, New York (2016). https://doi.org/10.1145/2993288.2993303
7. Ferreira, W., Baldassarre, M.T., Soares, S., Cartaxo, B., Visaggio, G.: A comparative study of model-driven approaches for scoping and planning experiments. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering. EASE 2017, pp. 78–87. ACM, New York (2017). https://doi.org/10.1145/3084226.3084258
8. Freire, M., et al.: A model-driven approach to specifying and monitoring controlled experiments in software engineering. In: Heidrich, J., Oivo, M., Jedlitschka, A., Baldassarre, M.T. (eds.) PROFES 2013. LNCS, vol. 7983, pp. 65–79. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39259-7_8

9. Garcia, R.E., Höhn, E.N., Barbosa, E.F., Maldonado, J.C.: An ontology for controlled experiments on software engineering. In: Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering (SEKE 2008), San Francisco, CA, USA, 1–3 July 2008, pp. 685–690. Knowledge Systems Institute Graduate School (2008)
10. Häser, F., Felderer, M., Breu, R.: An integrated tool environment for experimentation in domain specific language engineering. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. EASE 2016, pp. 20:1–20:5. ACM, New York (2016). https://doi.org/10.1145/2915970.2916010, http://doi.acm.org/10.1145/2915970.2916010
11. Hochstein, L., Nakamura, T., Shull, F., Zazworka, N., Basili, V.R., Zelkowitz, M.V.: An environment for conducting families of software engineering experiments. In: Software Development, Advances in Computers, vol. 74, pp. 175–200. Elsevier (2008). https://doi.org/10.1016/S0065-2458(08)00605-0
12. Jedlitschka, A., Ciolkowski, M., Pfahl, D.: Reporting experiments in software engineering. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering, pp. 201–228. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5_8
13. Johnson, P.M.: Requirement and design trade-offs in Hackystat: an in-process software engineering measurement and analysis system. In: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), pp. 81–90 (2007)
14. Karahasanoviæ, A., et al.: Collecting feedback during software engineering experiments. Empir. Softw. Eng. **10**(2), 113–147 (2005)
15. Lemos, O.A.L., Ferrari, F.C., Silveira, F.F., Garcia, A.: Development of auxiliary functions: should you be agile? An empirical assessment of pair programming and test-first programming. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 529–539, June 2012. https://doi.org/10.1109/ICSE.2012.6227163
16. Lemos, O.A.L., Ferrari, F.C., Silveira, F.F., Garcia, A.: Experience report: can software testing education lead to more reliable code? In: 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), pp. 359–369, November 2015. https://doi.org/10.1109/ISSRE.2015.7381829
17. Lemos, O.A.L., Silveira, F.F., Ferrari, F.C., Garcia, A.: The impact of software testing education on code reliability: an empirical assessment. J. Syst. Softw. **137**, 497–511 (2017)
18. Martin, R.C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design. Robert C. Martin Series, Prentice Hall, Boston (2017)
19. Mendonça, M.G., et al.: A framework for software engineering experimental replications. In: 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008), pp. 203–212, March 2008. https://doi.org/10.1109/ICECCS.2008.38
20. Neto, J.P., Scatalon, L.P.: ExpTool: a tool to conduct, package and replicate controlled experiments in software engineering. In: Proceedings of the International Conference on Software Engineering Research and Practice (SERP), p. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) (2014)

21. de Oliveira Neto, F.G., Torkar, R., Machado, P.D.L.: An initiative to improve reproducibility and empirical evaluation of software testing techniques. In: Proceedings of the 37th International Conference on Software Engineering - Volume 2. ICSE 2015, pp. 575–578. IEEE Press, Piscataway (2015). http://dl.acm.org/citation.cfm?id=2819009.2819106
22. Richards, M.: Software Architecture Patterns. O'Reilly Media Inc., Sebastopol (2015)
23. Scotto, M., Sillitti, A., Succi, G., Vernazza, T.: A non-invasive approach to product metrics collection. J. Syst. Archit. **52**(11), 668–675 (2006). https://doi.org/10.1016/j.sysarc.2006.06.010. Agile Methodologies for Software Production
24. Sjoeberg, D.I.K., et al.: A survey of controlled experiments in software engineering. IEEE Trans. Softw. Eng. **31**(9), 733–753 (2005). https://doi.org/10.1109/TSE.2005.97
25. de Souza, I.E., Oliveira, P.H.L., Bispo Junior, E.L., Inocencio, A.C.G., Parreira, Junior, P.A.: TESE - an information system for management of experimental software engineering projects. In: Proceedings of the Annual Conference on Brazilian Symposium on Information Systems: Information Systems: A Computer Socio-Technical Perspective - Volume 1. SBSI 2015, pp. 75:563–75:570. Brazilian Computer Society, Porto Alegre (2015). http://dl.acm.org/citation.cfm?id=2814058.2814147
26. Stolee, K.T., Elbaum, S.: Exploring the use of crowdsourcing to support empirical studies in software engineering. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM 2010, pp. 35:1–35:4. ACM, New York (2010). https://doi.org/10.1145/1852786.1852832
27. Torii, K., Matsumoto, K., Nakakoji, K., Takada, Y., Takada, S., Shima, K.: Ginger2: an environment for computer-aided empirical software engineering. IEEE Trans. Softw. Eng. **25**(4), 474–492 (1999). https://doi.org/10.1109/32.799942
28. Travassos, G.H., dos Santos, P.S.M., Mian, P.G., Neto, A.C.D., Biolchini, J.: An environment to support large scale experimentation in software engineering. In: 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008), pp. 193–202, March 2008. https://doi.org/10.1109/ICECCS.2008.30
29. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wessln, A.: Experimentation in Software Engineering. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29044-2
30. Yskout, K., Van Landuyt, D., Joosen, W.: Towards a platform for empirical software design studies. In: 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE), pp. 3–7 (2017). https://doi.org/10.1109/ECASE.2017.3