Software Quality Journal
https://doi.org/10.1007/s11219-021-09566-x

A dataset of regressions in web applications detected by end-to-end tests

Óscar Soto-Sánchez[1] · Michel Maes-Bermejo[1] · Micael Gallego[1] · Francisco Gortázar[1]

Accepted: 24 June 2021

Abstract
End-to-end tests present many challenges in the industry. The long-running times of these tests make it unsuitable to apply research work on test case prioritization or test case selection, for instance, on them, as most works on these two problems are based on datasets of unit tests. These ones are fast to run, and time is not usually a considered criterion. This is because there is no dataset of end-to-end tests, due to the infrastructure needs for running this kind of tests, the complexity of the setup and the lack of proper characterization of the faults and their fixes. Therefore, running end-to-end tests for any research work is hard and time-consuming, and the availability of a dataset containing regression bugs, documentation and logs for these tests might foster the usage of end-to-end tests in research works. This paper presents a) a dataset for this kind of tests, including six well-documented manually injected regression bugs and their corresponding fixes in three web applications built using Java and the Spring framework; b) tools for easing the execution of these tests no matter the infrastructure; and c) a comparative study with two well-known datasets of unit tests. The comparative study shows that there are important differences between end-to-end and unit tests, such as their execution time and the amount of resources they consume, which are much higher in the end-to-end tests. End-to-end testing deserves some attention from researchers. Our dataset is a first effort toward easing the usage of end-to-end tests in research works.

Keywords Dataset · Testing · End-to-end tests

✉ Michel Maes-Bermejo
michel.maes@urjc.es

Óscar Soto-Sánchez
oscar.soto@urjc.es

Micael Gallego
micael.gallego@urjc.es

Francisco Gortázar
francisco.gortazar@urjc.es

[1] Universidad Rey Juan Carlos, Móstoles 28933, Spain

1 Introduction

Empirical studies in software testing research require projects with known regressions that are the subject of this study. There are frameworks Just et al. (2014), infrastructures Do et al. (2005) and repositories Dallmeier and Zimmermann (2007); Spacco et al. (2005) in which different researchers have collected (or added manually) and documented existing bugs in different projects being available to researchers, allowing studies to be compared using the same bug dataset as a reference. The projects considered are usually open-source projects, which are more easily available, and they are usually libraries rather than applications. Some research datasets include the tests that reveal the different bugs. However, these tests are mostly unit tests, which in many research works can be a limitation, due to the specific characteristics that non-unit tests (integration, end-to-end or performance tests) have, such as their higher complexity when configured or the computational resources they require, in addition to their running time. The lack of datasets including non-unit tests makes researchers ignore them.

Non-unit tests are usually available as part of more complex projects (usually complete applications) that expose a front-end to interact with the user, a back-end to handle requests and a database to persist the information, as an example. These applications tend to have end-to-end tests (e2e), which usually impersonates the user using a browser that interacts with the application in order to ensure there are no regressions for the different use cases of the application. These tests allow more extensive studies on real-life applications. Some problems faced by the industry are based on the non-scalability of continuous integration systems in terms of dealing with an ever-increasing code base and tests, as Memon et al. (2017) report in Google, where every day 800,000 builds and 150 million test runs are performed in more than 13,000 projects. Most of the computational efforts are due to functional tests (integration and end-to-end tests) that require from minutes to hours to execute and much more computational resources (memory and CPU) than unit tests do.

We strongly believe that e2e tests should be subject of study, and practitioners are continuously expressing their frustration due to the several problems that this kind of tests have in their CI environments Liang et al. (2018). For instance, test prioritization might help in finding bugs sooner when the allotted test time is limited, for instance taking into account the time required by each test to execute. Other techniques might be explored as well. For instance, log analysis might indicate a likeliness of a long-running test to fail, by comparing the log with a successful one, and therefore, might interrupt the test to save some time Sarro (2018).

This paper presents a dataset of regression bugs which are revealed by e2e tests to support software testing research. A regression bug is defined by Nir et al. (2007) as *a bug which causes a feature that worked correctly to stop working after a certain event (system upgrade, system patching, daylight saving time switch, etc)*. The contributions of this paper are as follows:

- A repository comprised three complete web applications with multiple components (frontend, backend and a database). Each project consists of a Git repository with different branches that allow to explore the different changes the project has undergone as well as the buggy versions that have a documented regression bug. All regressions and fixes are properly tagged.
- E2e functional tests for each of the projects. Most of the tests use a browser to interact with the application, others use the API of the backend. In all the cases, the tests require

the whole application (the backend and a database) to be running. The specificities of how to run an application in order to run its e2e tests are one of the deterrents of using this kind of tests in research.
- Tools to build the projects and run the tests, using Docker[1] and Docker Compose. These tools allow researchers to reproduce any version of the code in a simple way to check the applications and their outputs (both in versions that work correctly and in those that contain an error).
- Extensive documentation of each regression bug accompanied by all resources that help to identify it; logs of the cases where the tests pass and fail, a visual diff-comparative of logs and videos of the e2e tests.
- A comparative study of unit and e2e tests in terms of running time, CPU and memory. Unit tests come from two popular datasets from the state-of-the-art in software testing research, and e2e tests come from the dataset proposed in this work.

This paper is an extension of a preliminary paper published in the 13th International Conference on the Quality of Information and Communications Technology (QUATIC, September 2020) Soto-Sánchez et al. (2020). This extension provides an extended state of the art, a more detailed description of the dataset itself, and a new section devoted to study how CPU usage, memory and running time metrics are different in this dataset (which uses e2e tests) with respect to other datasets of the literature (which uses only unitary tests). It also contains a detailed example using the test case prioritization (TCP) problem describing how different strategies are needed for building better solutions for the problem.

The paper is structured as follows: Previous work is presented in Sect. 2. The characteristics of the regression bugs are introduced in Sect. 3, and the dataset is described in detail in Sect. 4, including some use cases and a detailed example from the TCP problem. Section 5 presents a comparative study with other popular datasets containing exclusively unit tests. Threats to validity are presented in Sect. 6. Finally, Sect. 7 concludes the paper.

2 Related work

Datasets containing controlled faults and regressions are useful for doing controlled experiments where algorithms for different problems, like test prioritization, test selection, predictive analytics, among others, can be assessed. Furthermore, the availability of these datasets enables a fair comparison between different algorithms which is critical to understand how each algorithm compares with the others in their respective problems. These comparisons help researchers in understanding which techniques work better for a problem.

There have been several attempts at building a dataset of bugs and regression tests for them. As early as in 1994, Siemens Corporate Research conducted an experimental study that led to building a dataset containing a set of 130 errors Hutchins et al. (1994), introduced manually by the researchers in 7 projects written in C.

To the best of our knowledge, the first dataset of real bugs for use in software research is proposed by H. Do et al., *Software-Artifact Infrastructure Repository* Do et al. (2005), an infrastructure of 24 projects with documentation on 662 bugs (only 35 of them are real, the rest have been introduced by hand). In most projects, all the bugs are introduced

[1] https://www.docker.com/

by a single commit, being complicated to treat them individually. It is a dataset oriented to studies where the execution of the code is not a priority, oriented to make a static analysis of the code like in test case prioritization (TCP). However, the fact that all bugs are introduced in a single commit limits the applicability of certain techniques like code similarity, and definitely, it cannot be used for fault localization, where researchers look at the past commits of the project to determine where a bug was introduced.

Spacco et al. (2005) collected the bugs produced by students in a tool (Marmoset) where the code could be uploaded and checked on a server automatically. It includes eight different projects, carried out individually by 73 students, resulting in a total of 569 projects. They include not only the errors in the test, but also build errors of the project.

Bugs.jar Saha et al. (2018) is a large-scale dataset for research in automated debugging, patching, and testing of Java programs. It contains a total of 1,158 bugs and patches from eight large open-source Java projects.

The iBugs Dallmeier and Zimmermann (2007) project provides a repository of real bugs in Java projects. It contains 364 bugs, of which there is only one test that reveals the bug, and it includes mechanisms to get the corrected version of the bug, as well as its previous version for comparison. The dataset tools also allows the execution of the tests in both versions.

Defects4J Just et al. (2014) is an extensible framework which provides real bugs to enable reproducible studies. This framework contains 835 real bugs from 17 real-world open-source projects written in Java. Each bug included in their dataset contains information about the commit where the bug is fixed (which includes at least one test that reveals the bug), plus a failed commit to compare them. All bugs have their origin in the source code and are reproducible (both their failed and fixed versions), and the fix-commit does not include changes unrelated to the fix. The most recent version of the dataset includes a docker image to ease working with it.

BugsJS Gyimesi et al. (2019) is the first large benchmark of 453 real manually selected and validated JavaScript bugs from 10 popular server-side programs. Like Defects4J, it facilitates the reproducibility of the execution of the tests, specifically reproducing the environment from a Docker image.

Due to the increasing use of Python, datasets of this language have emerged for research. One of the most recent and relevant is BugsInPy Widyasari et al. (2020). This dataset is inspired by Defects4J and according to the authors follows a similar structure, including 493 bugs in 17 Python projects.

Another Python dataset, specifically oriented to research in Data Science, is Boa Meets Python Biswas et al. (2019), which gathers 1,558 Python projects available in GitHub; all of them focused on solving Data Science tasks like machine learning.

Although there are similarities between the dataset proposed in this paper and some datasets described here, such as facilitating the reproduction of tests using the Docker platform or labeling of buggy and fixed versions or a detailed documentation of errors, none of these datasets have e2e tests. Therefore, this limits the research of specific techniques that could work well for e2e tests. Furthermore, some techniques cannot be applied to unit tests, like analytics on long-running tests with rather long logs. Our dataset contributes to this area by focusing specifically on e2e tests, easing the execution of those tests by providing a tool to run them, and carefully design the faults and their fixes. The authors hypothesize that this lack of e2e tests in state-of-the-art datasets is due to these tests requiring more configuration (deployment of the whole application and its dependencies, like databases, and the use of external browsers) and resource consumption (running time, CPU and memory) than unit tests, making it a challenge to reproduce them correctly. Our dataset simplifies all this configuration by providing a tool to run all the tests easily.

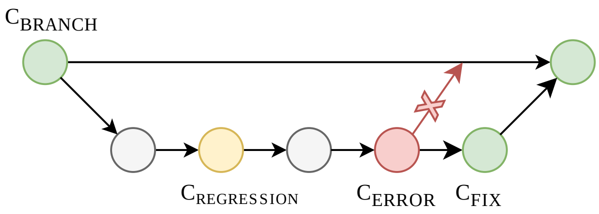


Fig. 1 A simplified example of a commit history

3 Generation of the regression bugs

The main difference between the dataset described in this work and the ones mentioned in Sect. 2 is the use of complete applications which are made up of different components: a client side (frontend), a server side (backend) and a database, along with e2e tests (in contrast to the unit tests used by other existing datasets). This section describes the methodology used to generate the bugs as well as their characteristics. The main objective is to make the process as close as possible to how actual projects introduce new regression bugs.

To ensure bugs are representative of these web applications, we considered bugs in both the frontend and the backend. Frontend bugs are rather difficult to track because the frontend usually log the problem in the browser. Unless the developer retrieves these logs, they will not be available. In the dataset, these logs are available, as they were retrieved when running the tests. Some bugs are related to security concerns (pages a user should not see, but that he or she can navigate to) to consider as well situations that might compromise the security of the application.

3.1 Methodology

The approach followed to introduce faults is as follows: a branch per feature development model is assumed, where a new branch is created for each feature, and all the work up until the feature is finished is done in the branch. At the end, the branch is merged with the main branch of the project. Therefore, this branch includes multiple commits, simulating the changes that would occur in a real project when adding a new feature. In the dataset, one of these commits in the branch will introduce a regression. Some more commits after, the new functionality will be tested by launching the tests. At this point, some tests will reveal the regression, that will then be fixed and the new feature will be merged to the main branch. These steps are explained in detail below and are graphically depicted in Fig. 1, where each node represents a commit (a set of changes in the project history).

First a feature branch will be created, namely (C_{branch}). This branch is created in order to add a new feature to the application. At the point of branch creation, C_{branch}, all tests pass. These tests are the same that will detect in the future the regression.

Once the new branch is created, development of the feature starts, and as a result several commits are added to it. One of the commits, $C_{regression}$, introduces a regression bug. Therefore, at least one test would fail at that commit, because all regression bugs introduced are caught by at least one test.

However, in the context of continuous integration (CI) it is common that these tests are not executed at each commit, so it might be the case that the regression is not detected until we try to merge the branch with the main branch. Most CI servers will fail the merge, because the tests did not pass. Let us assume the commit at which the error is detected is C_{error}.

Once tests fail at C_{error}, developers develop a fix commit, C_{fix}, that solves the bug. At this C_{fix} commit, all tests pass again and the feature branch can be successfully merged with the main branch.

This process was followed by the authors when introducing regression bugs in the three web applications of the dataset. As a result, the corresponding branches are quite similar to what happens in real projects, but with a proper characterization of how and when the bug was introduced, at which commit it was fixed and the specific tests that detect the regression. Authors think this characterization is important in order to ensure the dataset can be used for different problems.

3.2 Properties

When generating regression bugs, it was ensured that the following properties apply to all of them:

- **The bug is reproducible** Any regression bug must be reproducible, i.e., running the tests always generates the same output. Notice that this is a decision by the authors and does not limit the extensibility of the dataset to include as well flaky tests. These are perfectly possible, but out of the scope of this work.
- **The bug is related to source code** The bug cannot be related to the build system, the configuration, the environment in which it runs, or the test files. Instead, the root cause of the bug must be related to (and can be traced to) a change in the source code of the application.
- **The change is realistic** The commit in which the bug is introduced must contain more changes than the bug itself. The branch where this bug is placed must contain more commits with different changes in the application. This is to imitate the natural changes that occur in the application and in which a regression usually appears.
- **The tests are end-to-end** The tests that detect the regression are functional tests that use a browser to interacts with the application as a user would do.

4 Dataset of regression bugs

4.1 Subject applications

For the realization of our dataset, three applications from the students of the subject *Development of Web Applications* from the Software Engineering degree at Rey Juan Carlos University were selected. These projects have been built as complete and functional applications, getting as close as possible to how an application would be developed in the industry. All the applications consist of a backend written in Java using the Spring framework and a frontend, and the three of them require a MySQL database to store contents.

- **Webapp-1 (social network)** This application consists of a backend developed in Java with Spring and a frontend developed in Angular with TypeScript. This application is a social network of films, series and books.

Table 1 Non-commenting source statements (NCSS) metrics

NCSS	Webapp-1	Webapp-2	Webapp-3
Backend (Spring)	6,097	4,890	3,832
Frontend (Angular)	26,170	5,452	0
Total	32,267	10,342	3,832

- **Webapp-2 (online courses)** This application consists of a backend developed in Java with Spring, and a frontend developed in Angular with TypeScript. This application is a platform for online courses.
- **Webapp-3 (library)** This application consists of a backend developed in Java with Spring and a frontend developed with Moustache (a template engine) using Spring. This application is the web page of a library.

Both Webbapp-1 and Webapp-2 are *Single Page Applications* (SPA), whereas Webapp-3 is a MVC application. The applications presented above are complex to test because they are composed of several parts each running in its own process: a backend, a frontend running in the browser, and a database. All the parts that make up the application must be up and running before any test can be launched (this process is a deterrent on the use of e2e tests in academia and is further discussed in Section 4.3). To give an idea of the size of the three web applications, Table 1 shows the non-commenting source statements (NCSS) metrics for each of them. As shown in the table, they are applications with an important size in terms of their lines of code.

4.1.1 Webapp-1: social network web application

This application is a social network of multimedia content (films, series and books). It allows users to assign a score to the different multimedia content. It is also possible to comment on the multimedia content to provide the user's opinion on the different films, series or books. Users can create lists with the multimedia content in order to keep track of the things that have been seen or liked the most. The application also has a series of graphics with the most valued and most viewed multimedia content in its different sections (films, series and books). The home page of the application is shown in Fig. 2.

This application has three different roles:

- **User role** will be able to rate the multimedia content, comment on the multimedia content and create and manage their lists.
- **Moderator role** adds on top of the permissions granted by the user role the capability to moderate user comments.
- **Administrator role** adds on top of the moderator role the capability to add, edit and delete the different multimedia content contained in the page.

4.1.2 Webapp-2: online courses web application

This application is an online course platform. Users can register for the different courses offered on the platform. Once a user is enrolled in a course, a list with each of the subjects of the course will be accessible. Within each of the subjects, the teachers will upload the

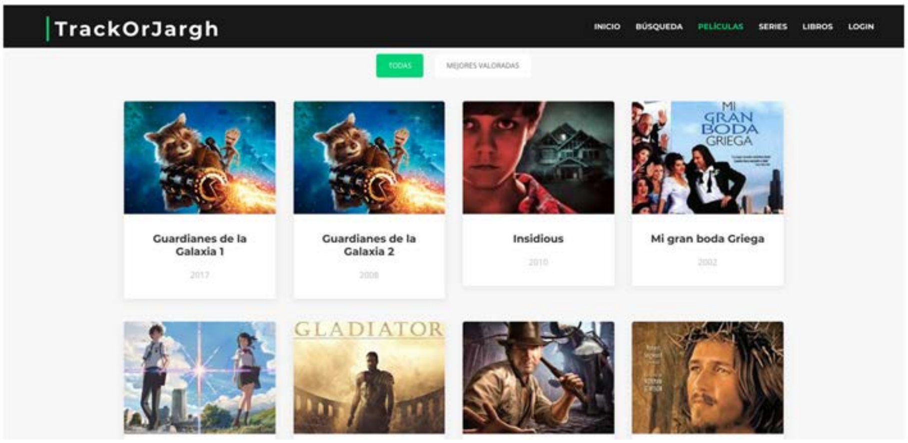


Fig. 2 Webapp-1 home page

necessary material for the students to study the subject. Teachers will also add assignments that the students will have to do to pass the subject. Student passing all the subjects of a course, are given a certificate of completion of the course. The home page of the application is shown in Fig. 3.

This application has three different roles:

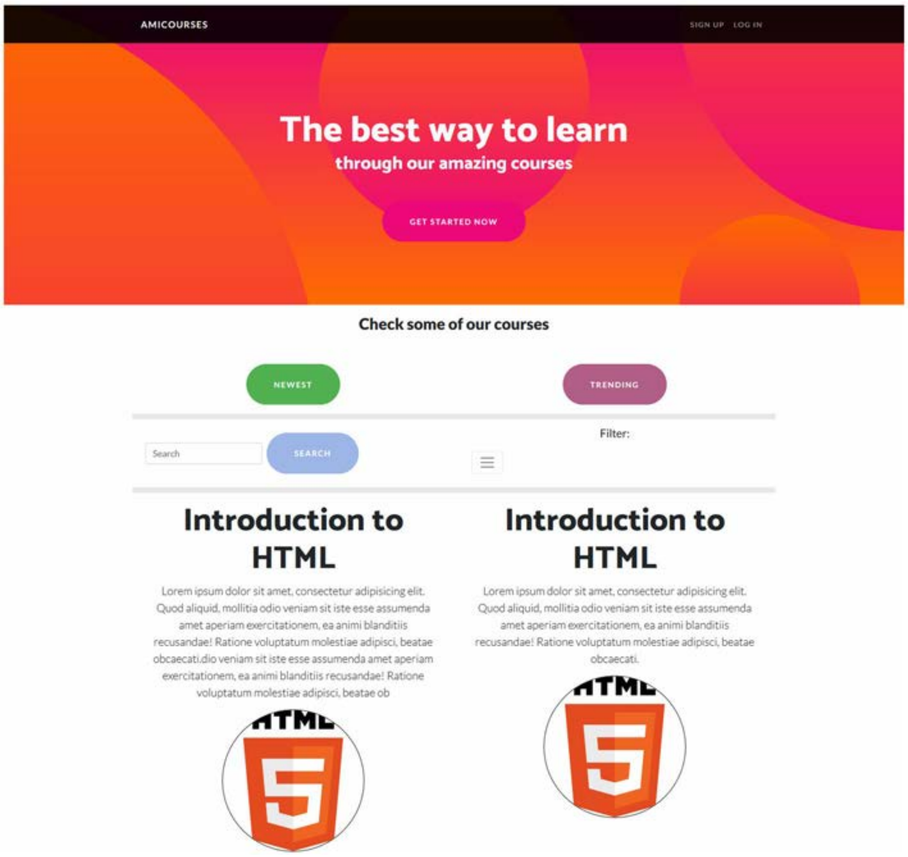
- **User role** will allow students to register for the different courses offered and obtain a certificate of completion.
- **Teacher role** will be given to teachers to manage the courses by uploading the necessary material, adding assignments and evaluating the students enrolled on the course.
- **Administrator role** will be given to administrators to manage the creation, edition and deletion of the different courses. Within the courses, administrators assign the teachers and manage the subjects taught. They are also responsible for creating the platform's teacher accounts.

4.1.3 Webapp-3: library web application

This application is a library platform. Users can search resources in the library (books or journals) and book them for later collection in person. They can as well specify the day in which the resources they have will be returned. This application's home page is shown in Fig. 4.

The application has two different roles:

- **User role** enabling users to book and return the different resources offered by the library.
- **Administrator role** enabling administrators to add, edit and delete the different resources of the library and create, modify and delete users. This role will also allow administrators to manage the return of material and assign fines if a resource is not returned on time.



Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quod aliquid, mollitia odio veniam sit iste esse assumenda amet aperiam exercitationem, ea animi blanditiis recusandae! Ratione voluptatum molestiae adipisci, beatae obcaecati.dio veniam sit iste esse assumenda amet aperiam exercitationem, ea animi blanditiis recusandae! Ratione voluptatum molestiae adipisci, beatae ob

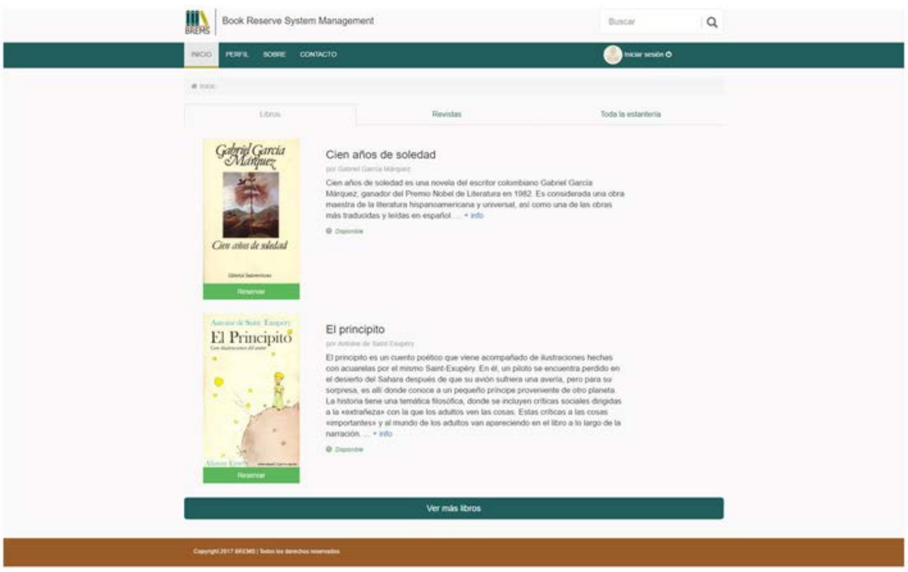
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quod aliquid, mollitia odio veniam sit iste esse assumenda amet aperiam exercitationem, ea animi blanditiis recusandae! Ratione voluptatum molestiae adipisci, beatae obcaecati.

Fig. 3 Webapp-2 home page

4.2 Anatomy of introduced regression bugs

The presented dataset contains three projects (applications) with a total of six regression errors, the exact number of regressions that each application contains is shown in Table 2. These applications were selected from those developed by the students of the subject Development of Web Applications, at the Software Engineering degree at Rey Juan Carlos University, and they simulate a real application.

Regression errors presented in this section were introduced by hand by the authors of the paper, using the methodology described in section 3. Figure 5 shows how a regression was introduced in Webapp-1. Each node in the image represents a commit that belongs to a branch where a feature is being developed. The image shows two branches, namely *fixed-footer* and *refactor-index-charts*. Each of the commits may or may not contain a regression. The last commit of the branch, the one merged into the main (master) branch, contains the regression bug fixed. The two branches shown in the figure correspond to the 2 regressions added in the Webapp-1; that is, each branch contains a commit that introduces a regression, and this regression is fixed in the last commit of the branch.



Cien años de soledad

por Gabriel García Márquez

Cien años de soledad es una novela del escritor colombiano Gabriel García Márquez, ganador del Premio Nobel de Literatura en 1982. Es considerada una obra maestra de la literatura hispanoamericana y universal, así como una de las obras más traducidas y leídas en español. ... + info

Disponible

El principito

por Antoine de Saint Exupéry

El principito es un cuento poético que viene acompañado de ilustraciones hechas con acuarelas por el mismo Saint-Exupéry. En él, un piloto se encuentra perdido en el desierto del Sahara después de que su avión sufriera una avería, pero para su sorpresa, es allí donde conoce a un pequeño príncipe proveniente de otro planeta. La historia tiene una temática filosófica, donde se incluyen críticas sociales dirigidas a la «extrañeza» con la que los adultos ven las cosas. Estas críticas a las cosas «importantes» y al mundo de los adultos van apareciendo en el libro a lo largo de la narración. ... + info

Disponible

Copyright 2017 BREMS | Todos los derechos reservados

Fig. 4 Webapp-3 home page

All regression bugs introduced in the different applications of the dataset are presented below, and the tests that detect the regression bugs are presented as well. How to run the tests is presented in the resource [2] and discussed further in Section 4.3.

– **Webapp-1 Regression 1** In this regression, the bug is introduced in a frontend component, that is, a component that runs as part of the UI in the browser. The error is due to a malfunction of an element in the HTML of the page. This element checks if a non-existent variable contains a Boolean value. Due to the fact that the variable is not defined, an error occurs that prevents the creation of a new list to be presented to the user. In order to locate this error, it is necessary to see the video of the execution (browser windows are recorded and the videos are available in the dataset). In the video recording[3] showing the regression it is shown how the add button does not work and in the video recording[4] of the fixed regression it is shown how the button works again. The component in which the bug is found is *profile.html*, which is part of the frontend. The test detecting the regression is *TestE2EFront*.

Table 2 Applications and number of regression bugs available in our dataset

Application	# of regressions
Webapp-1	2
Webapp-2	3
Webapp-3	1

[2] https://git.io/JOesQ
[3] https://git.io/Jk9ai
[4] https://git.io/Jk98O

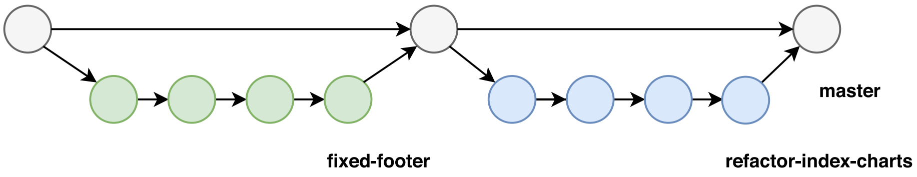
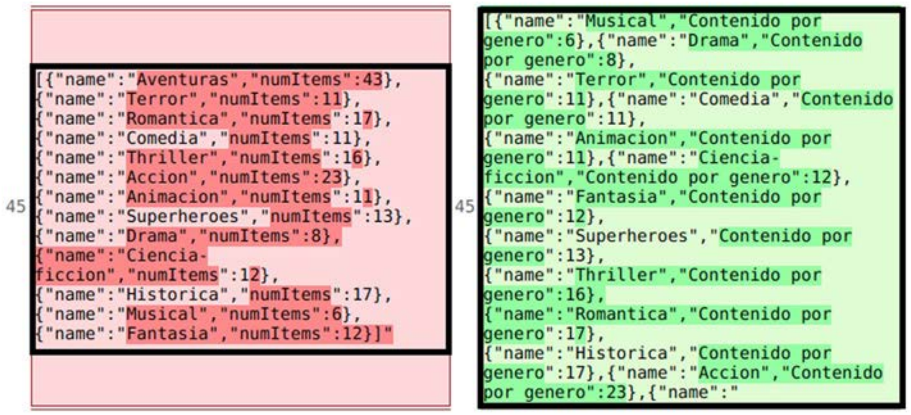


Fig. 5 In this figure, we can see the webapp-1 git history graph

- **Webapp-1 Regression 2** The component containing the bug in this regression is part of the backend. The error is due to an erroneous count in the number of genres of the different multimedia content (films, series and books). The method in charge of this operation contains a bug, and the backend API returns a wrong number. The error can be caught by looking at the logs returned by the backend. Figure 6 shows a comparison of the logs recorded for a version of the application with the regression (left) and a version without the regression (right). The figure shows how the API returns a different number in both cases (red and green squares in the image). This information is available in the repository[5]. The component in which the bug is found is *ApiGenderController.java*, which is part of the backend. This fault is detected by the test *TestAPIRestTemplate*.
- **Webapp-2 Regression 1** The component containing the bug in this regression is part of the backend. The error is due to poor management of user permissions. This bad management allows any user to see another user's profile when this should not be allowed. The error becomes more evident by having a look at the backend and frontend logs. Figure 7 shows how the API returns the profile of a user. That should not happen, as according to the logs the user is not logged in. By comparing the run of the test in the regression commit (red square) and the fixed commit (green square), it is possible to

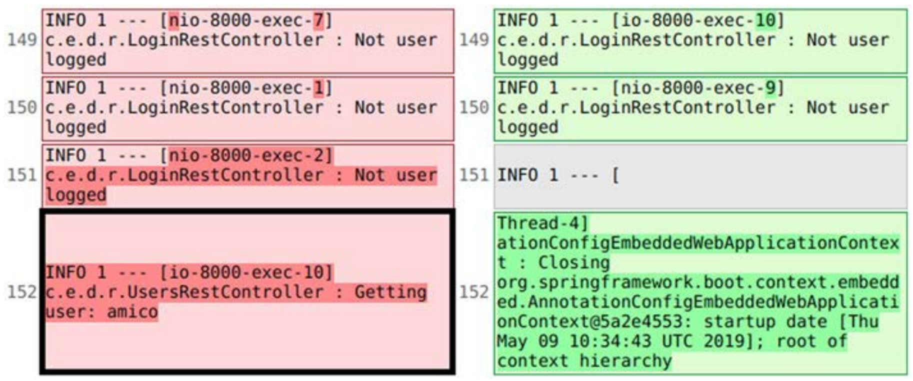


```
[{"name":"Aventuras","numItems":43},
{"name":"Terror","numItems":11},
{"name":"Romantica","numItems":17},
{"name":"Comedia","numItems":11},
{"name":"Thriller","numItems":16},
{"name":"Accion","numItems":23},
{"name":"Animacion","numItems":11},
{"name":"Superheroes","numItems":13},
{"name":"Drama","numItems":8},
{"name":"Ciencia-
ficcion","numItems":12},
{"name":"Historica","numItems":17},
{"name":"Musical","numItems":6},
{"name":"Fantasia","numItems":12}]"
```

```
[{"name":"Musical","Contenido por
genero":6},{"name":"Drama","Contenido
por genero":8},
{"name":"Terror","Contenido por
genero":11},{"name":"Comedia","Contenido
por genero":11},
{"name":"Animacion","Contenido por
genero":11},{"name":"Ciencia-
ficcion","Contenido por genero":12},
{"name":"Fantasia","Contenido por
genero":12},
{"name":"Superheroes","Contenido por
genero":13},
{"name":"Thriller","Contenido por
genero":16},
{"name":"Romantica","Contenido por
genero":17},
{"name":"Historica","Contenido por
genero":17},{"name":"Accion","Contenido
por genero":23},{"name":"
```

Fig. 6 Webapp-1 Regression Bug 2

5 https://git.io/Jk9rY



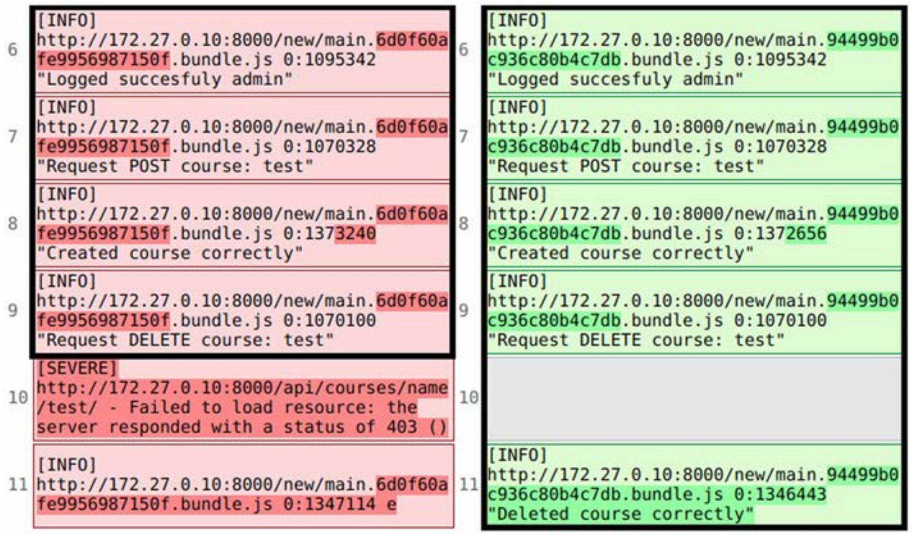
```
INFO 1 --- [nio-8000-exec-7]
c.e.d.r.LoginRestController : Not user
logged
INFO 1 --- [nio-8000-exec-1]
c.e.d.r.LoginRestController : Not user
logged
INFO 1 --- [nio-8000-exec-2]
c.e.d.r.LoginRestController : Not user
logged
INFO 1 --- [io-8000-exec-10]
c.e.d.r.UsersRestController : Getting
user: amico
```

```
INFO 1 --- [io-8000-exec-10]
c.e.d.r.LoginRestController : Not user
logged
INFO 1 --- [nio-8000-exec-9]
c.e.d.r.LoginRestController : Not user
logged
INFO 1 --- [
Thread-4]
ationConfigEmbeddedWebApplicationContex
t : Closing
org.springframework.boot.context.embedd
ed.AnnotationConfigEmbeddedWebApplicati
onContext@5a2e4553: startup date [Thu
May 09 10:34:43 UTC 2019]; root of
context hierarchy
```

Fig. 7 Webapp-2 Regression Bug 1

see how the version of the application with the regression is returning a user profile. This information is available in the repository[6]. The component containing the bug is *RestSecurityConfig.java*, which is part of the backend. The test revealing this fault is *TestE2EFront*.

– **Webapp-2 Regression 2** The component containing the bug in this regression is part of the backend. The error is due to poor management of user permissions. This bad management of permissions prevents the administrator user from being able to delete courses registered on the platform. In order to locate this error, it is necessary to be able to see the logs offered by the browser, which are not usually available. However, in the dataset these logs have been extracted and stored as well. Figure 8 shows how the API



```
[INFO]
http://172.27.0.10:8000/new/main.6d0f60a
fe9956987150f.bundle.js 0:1095342
"Logged succesfuly admin"
[INFO]
http://172.27.0.10:8000/new/main.6d0f60a
fe9956987150f.bundle.js 0:1070328
"Request POST course: test"
[INFO]
http://172.27.0.10:8000/new/main.6d0f60a
fe9956987150f.bundle.js 0:1373240
"Created course correctly"
[INFO]
http://172.27.0.10:8000/new/main.6d0f60a
fe9956987150f.bundle.js 0:1070100
"Request DELETE course: test"
[SEVERE]
http://172.27.0.10:8000/api/courses/name
/test/ - Failed to load resource: the
server responded with a status of 403 ()
[INFO]
http://172.27.0.10:8000/new/main.6d0f60a
fe9956987150f.bundle.js 0:1347114 e
```

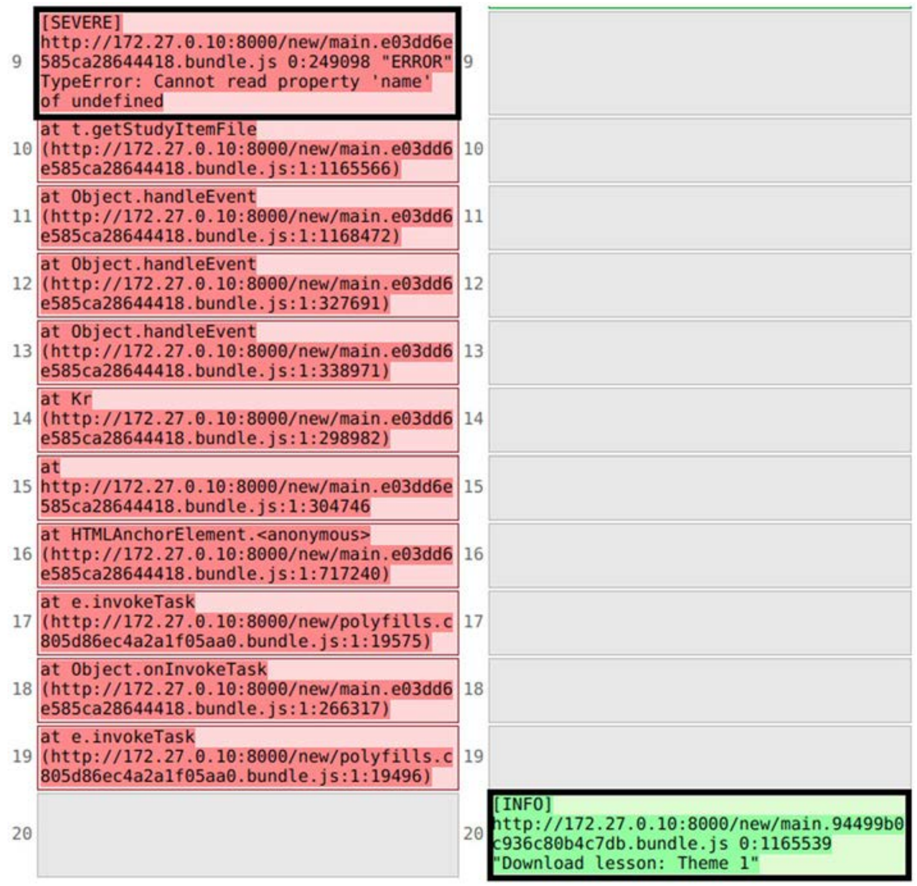
```
[INFO]
http://172.27.0.10:8000/new/main.94499b0
c936c80b4c7db.bundle.js 0:1095342
"Logged succesfuly admin"
[INFO]
http://172.27.0.10:8000/new/main.94499b0
c936c80b4c7db.bundle.js 0:1070328
"Request POST course: test"
[INFO]
http://172.27.0.10:8000/new/main.94499b0
c936c80b4c7db.bundle.js 0:1372656
"Created course correctly"
[INFO]
http://172.27.0.10:8000/new/main.94499b0
c936c80b4c7db.bundle.js 0:1070100
"Request DELETE course: test"
[INFO]
http://172.27.0.10:8000/new/main.94499b0
c936c80b4c7db.bundle.js 0:1346443
"Deleted course correctly"
```

Fig. 8 Webapp-2 Regression Bug 2

[6] https://git.io/Jk9Kl

does not allow an administrator to delete a course. This is clearer by looking at the logs of the browser when running the test on the regression commit (red square) and at the commit that fixes the regression (green square). This information is available in the repository[7]. The component in which the bug is found is *RestSecurityConfig.java* which is part of the backend. The test revealing this fault is *TestE2EFront*.

– **Webapp-2 Regression 3** The component containing the bug in this regression is part of the frontend. The error is due to a malfunction of an element in the HTML. This element misspells a function name. Due to the fact that the name of the method does not exist, there is an error that prevents downloading the course's content. In order to locate this error it is necessary to retrieve the logs offered by the browser. These logs have been extracted and included in the dataset. Figure 9 shows how the Angular framework reports a property not found error. This bug is easy to catch by looking at the browser logs (not usually available). In the figure, the logs of the version of the application with

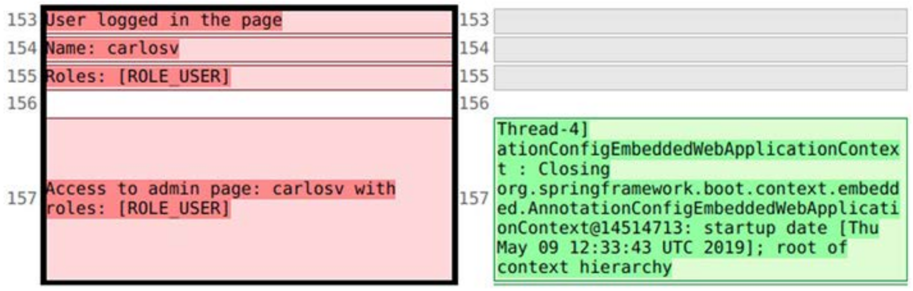


```
9  [SEVERE]
   http://172.27.0.10:8000/new/main.e03dd6e
   585ca28644418.bundle.js 0:249098 "ERROR"
   TypeError: Cannot read property 'name'
   of undefined
10 at t.getStudyItemFile
   (http://172.27.0.10:8000/new/main.e03dd6
   e585ca28644418.bundle.js:1:1165566)
11 at Object.handleEvent
   (http://172.27.0.10:8000/new/main.e03dd6
   e585ca28644418.bundle.js:1:1168472)
12 at Object.handleEvent
   (http://172.27.0.10:8000/new/main.e03dd6
   e585ca28644418.bundle.js:1:327691)
13 at Object.handleEvent
   (http://172.27.0.10:8000/new/main.e03dd6
   e585ca28644418.bundle.js:1:338971)
14 at Kr
   (http://172.27.0.10:8000/new/main.e03dd6
   e585ca28644418.bundle.js:1:298982)
15 at
   http://172.27.0.10:8000/new/main.e03dd6e
   585ca28644418.bundle.js:1:304746
16 at HTMLAnchorElement.<anonymous>
   (http://172.27.0.10:8000/new/main.e03dd6
   e585ca28644418.bundle.js:1:717240)
17 at e.invokeTask
   (http://172.27.0.10:8000/new/polyfills.c
   805d86ec4a2a1f05aa0.bundle.js:1:19575)
18 at Object.onInvokeTask
   (http://172.27.0.10:8000/new/main.e03dd6
   e585ca28644418.bundle.js:1:266317)
19 at e.invokeTask
   (http://172.27.0.10:8000/new/polyfills.c
   805d86ec4a2a1f05aa0.bundle.js:1:19496)
20

9
10
11
12
13
14
15
16
17
18
19
20 [INFO]
   http://172.27.0.10:8000/new/main.94499b0
   c936c80b4c7db.bundle.js 0:1165539
   "Download lesson: Theme 1"
```

Fig. 9 Webapp-2 Regression Bug 3

[7] https://git.io/Jk9iE



```
User logged in the page
Name: carlosv
Roles: [ROLE_USER]

Access to admin page: carlosv with
roles: [ROLE_USER]
```

```
Thread-4]
ationConfigEmbeddedWebApplicationContex
t : Closing
org.springframework.boot.context.embedd
ed.AnnotationConfigEmbeddedWebApplicati
onContext@14514713: startup date [Thu
May 09 12:33:43 UTC 2019]; root of
context hierarchy
```

Fig. 10 Webapp-3 Regression Bug 1

the regression (red square) contains an error, which does not appear on the logs of the version without the regression (green square). This information is available in the repository[8]. The component in which the bug is found is *component.html* which is part of the frontend. This fault is detected by the test *TestE2EFront*.

– **Webapp-3 Regression 1** The component containing the bug in this regression is part of the backend. The error is due to poor management of user permissions. This bad management allows any user to access the administration panel and make changes to the platform as if they were an administrator. The bug can be caught by having a look at the backend and frontend logs. Figure 10 shows how the API allows a user without an administrator role to access the administration panel. By comparing the logs of a version with the regression (red square) and a fixed version (green square) it is possible to see how in the version with the regression the user could log in and access the admin page. This information is available in the repository[9]. The component in which the bug is found is *SecurityConfiguration.java*, which is part of the backend. The fault is revealed by the test *TestE2EFront*.

4.3 Running end-to-end tests

Running end-to-end tests is not trivial. In order to run them, first the whole application must be running. The application is generally comprised of a backend that provides a REST API to the fronted, and a database where the application information will be stored. Finally, the frontend, which runs in the browser, will connect to the backend API to interact with the application.

Although these web applications might have a different way of being run, as a general rule it is not usually specified how to run the different components. For instance, usually little or nothing at all is said about the database: which version to run or which configuration to apply. This is changing slowly with the advent of Docker containers; nevertheless, there is still a need to understand which containers to launch or which parameter to pass in. In the dataset, we made an effort to package all applications using Docker containers

[8] https://git.io/Jgreen
[9] https://git.io/Jk9PE

```
$ cd
$ git clone --recursive https://github.com/e2e-tests-dataset/e2e-tests-
    dataset
```

Listing 1 Cloning the e2e-tests-dataset projectCloning the e2e-tests-dataset project

```
$ cd $HOME/e2e-tests-dataset/Application-2/webapp-2
$ git checkout regression-1
$ cd $HOME/e2e-tests-dataset/Application-2/webapp-2/AMICOServer
$ mvn clean package -B -Dskip.test
$ cd $HOME/e2e-tests-dataset/Application-2/webapp-2/Angular
$ npm install --unsafe
```

Listing 2 Compilation of application webapp-2 from branch regression-1

```
$ cd $HOME/e2e-tests-dataset/Application-2/webapp-2/AMICOServer
$ java -jar target/AMICOServer-*.jar > /tmp/backend-build.log 2>&1 &
$ cd $HOME/e2e-tests-dataset/Application-2/webapp-2/Angular
$ npx ng serve --host 0.0.0.0 > /tmp/frontend.log 2>&1 &
```

Listing 3 Execution of the backend and the frontend of the application webapp2

```
$ cd $HOME/e2e-tests-dataset/Application-2/webapp-2/AMICOServer
$ mvn -Dtest=TestE2EFront #checkShowProfile test
```

Listing 4 Launch the webapp-2 application tests

so that the different parts of the application (backend, frontend, database, web browser and test) can be run with a single command.

As an example of the complexity of running e2e tests, this section first shows how to run end-to-end tests by hand and then how the same test can be run using the tool provided in the dataset, simplifying the run of this kind of tests.

4.3.1 Running tests manually

For this example, webapp-2 and the regression-1 of the dataset will be used. The example run was performed using an *Ubuntu 20.04 OS*, and all commands were launched in the *home* folder. It is worth noticing that this is the process that a researcher would have to follow in order to just run the tests. Collecting detailed information like running times, memory and CPU consumption, or logs of the different components require additional efforts.

First, the project *e2e-tests-dataset* must be made available in the system, by cloning its Git repository. For this *Git 2.17.1* or higher must be installed. Listing 1 shows the commands required for cloning the project.

Second, we move to the folder of the application (webapp-2) and change the branch to the branch containing the regression (regression-1). Now, the application is built. As this is an SPA application, both the backend and the frontend need to be built. For the compilation, *Java 8*, *Maven 3.6.0*, *NodeJS 8.17.0* and *npm 6.13.4* or higher are needed. Note

```
$ docker run --rm -v /tmp/e2e-dataset/logs:/home/dataset/logs \
  codeurjc/e2e-dataset webapp-2 regression-1
```

Listing 5 Docker command for launching the webapp-2 application tests in the regression-1 branch

```
$ docker run --rm -v /tmp/e2e-dataset/logs:/home/dataset/logs \
  codeurjc/e2e-dataset webapp-2 regression-fixed-1
```

Listing 6 Docker command for launching the tests of the webapp-2 application in the regression-fixed-1 branch

how the versions matter, and the researcher would have to know which specific versions to use. Something that is not always clearly stated in the project documentation. Listing 2 shows the commands required for checking out the branch regression-1 and compiling the webapp-2 application.

Third, in order to launch the tests, first the backend and frontend of the application should be executed. For running the different parts of the applications, a *MySQL 5.7* database, *Java 8* and *NodeJS 8.17.0* or higher are needed. Notice how standard output is redirected to a log file in order to have the logs of the different components of the application available for later inspection. Listing 3 shows the commands required for executing both the backend and the frontend of Webapp-2 application.

Fourth, and last, when the application is up, the tests are run. To do so, a *Chrome 75.0.3770.80* web browser or higher must be available in the system, as the tests will start a browser and interact with the application through it. Listing 4 shows the commands required for launching the tests of the webapp-2 application.

Once the tests have been run, if some of them fail, researchers would be able to check the logs of the application reviewing the files *'/tmp/backend-build.log'* for the backend logs and *'/tmp/frontend.log'* for the fronted logs. Additionally, the output of the test is available in the terminal where they were run. Notice that in this case, the browser logs are not available. Getting these logs is not as simple as redirecting the standard output of the browser to a file. In order to retrieve these logs, the test itself needs to query them through the library used to control the browser (Selenium in the dataset). Usually, researchers are not interested in modifying in any way the original test code.

To run the application *webapp-2* from a commit where the regression has been repaired (i.e., without the regression), all the previous steps but the first one need to be run again, changing the branch to *'regression-fixed-1'* in the second step.

More details on how to manually run the different tests available in the dataset can be found in the repository [10].

4.3.2 Running end-to-end tests using the Docker image provided in the dataset

Within the dataset, we provide a Docker image that simplifies the run of e2e tests from the dataset. Using this image, there is no need to have development tools or a web browser installed locally, which avoids for any problem arising from incorrect versions of the tools, or the unavailability of the corresponding web browser. For this example, the *webapp-2*

[10] https://git.io/JOesQ

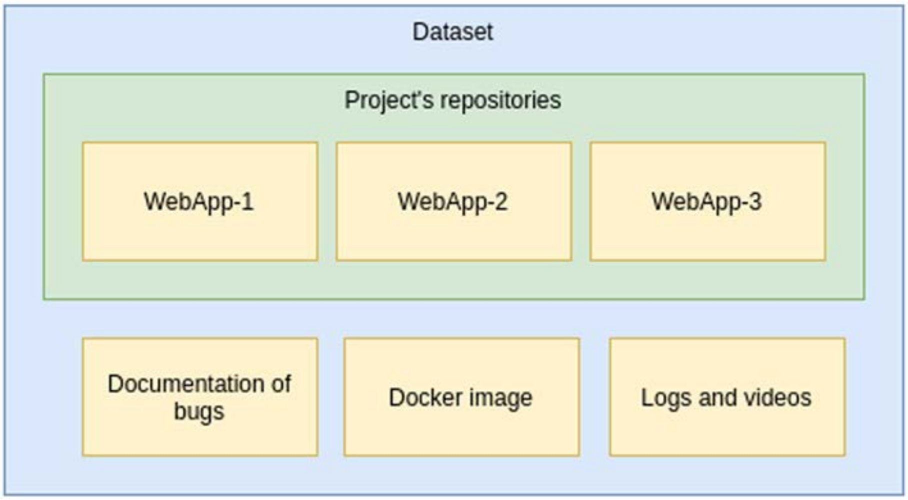


Fig. 11 Conceptual model of e2e dataset

and the *regression-1* of the dataset will be used. The whole application and the test were run in an *Ubuntu 20.04 OS*, and all commands were launched in the *home* folder.

In this case, the only requirement is to have a *Docker 19.03* or higher installed. Everything can be accomplished with a single command as shown in Listing 5.

Once the tests finish, all the logs will be made available in the folder *'/tmp/e2e-dataset/logs'*. Notice that the researcher might choose any other folder to mount in the container.

Listing 6 shows the command for running the repaired version of the application (the one without the regression).

To find out how to run the different tests available in the dataset using the Docker image, we reference the reader to the dataset repository.

4.4 Dataset contents

The dataset contains three web applications. For each application, there are some regression bugs and tests properly documented. Logs, videos and detailed information of the commit where the regression is introduced, as well as the commit where the regression is fixed, are included. Any researcher is able to obtain the same logs when running the tests on the different highlighted commits. For this purpose, a Docker image is provided to allow the deployment and execution of the tests in a simple way. A conceptual model of the dataset is presented in Fig. 11.

Each of the artifacts of the dataset is detailed below:

- **Source code** The dataset provides a git code repository with the source code of the three applications used for the creation of the regression bugs.
- **Document with bugs** A document is provided describing the regression bugs, explaining how they work, where the regression bug was introduced and the test that detects the bug.

- **Logs with the correct execution** The dataset contains text files with the correct run output, including backend logs, frontend logs (logs exposed by the frontend part of the application on the browser) and test logs.
- **Logs with the regression bug execution** The dataset provides text files with the regression bug run output, including backend logs, frontend logs and test logs.
- **Logs comparison** The dataset provides a comparison of the logs obtained for the correct run and regression bug run. For this comparison, we used the library diff-math-patch[11], which makes use of the Myer's algorithm Myers (1986).
- **Videos** The dataset contains two videos of the test, the first video corresponds to a correct run of the application (the one without the regression) and the second video corresponds to the regression bug run.
- **Docker image** The dataset provides a docker image with the projects and all dependencies that are needed in order to run the application, along with a script to build the image for any commit in history.

In order to be able to collect all the information related to the application run, we have used the ElasTest tool Gortazár et al. (2017), (2018); Bertolino et al. (2018) which allowed the authors to execute tests, capture the logs and the videos, and produce the log comparisons. ElasTest provided as well the browser logs that would have been difficult to obtain without modifying the test cases.

The dataset is public and available on this GitHub repository:

https://github.com/e2e-tests-dataset/e2e-tests-dataset

The rest of this section provides a detailed explanation of the different artifacts of the dataset.

4.4.1 Source code

Each of the three projects is provided under its own folder in the root folder of the dataset repository. The actual source code of the application is provided as a git submodule and is stored in a separate Git repository. Along with the source code, the project's folder contains a regression folder that includes a subfolder for each regression in the project.

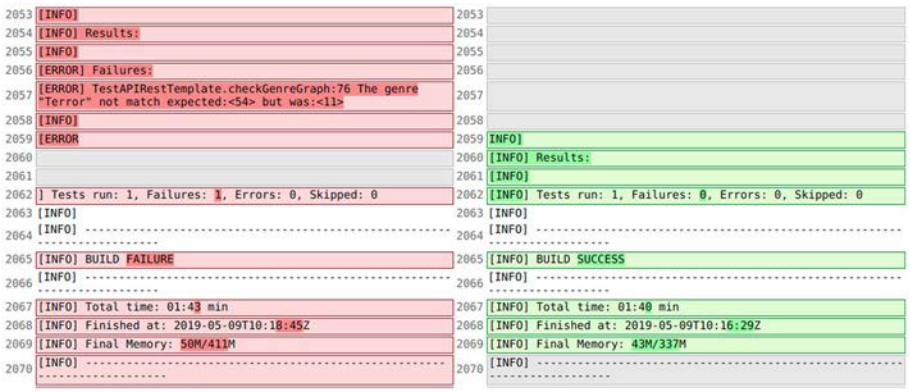
Regression's folders contain the detailed documentation of the regression along with the related artifacts, as described below.

4.4.2 Document with bugs

Within each regression folder a `README.md` document is provided, describing in detail the regression. Specifically, this document contains:

- The branch where the regression is introduced.
- The link to the branch within the Git repository.
- The test that reveals the regression.
- The link to this test.
- The files that changed in the branch.
- The links to those files.

[11] https://github.com/google/diff-match-patch



```
[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR] TestAPIRestTemplate.checkGenreGraph:76 The genre
"Terror" not match expected:<54> but was:<11>
[INFO]
[ERROR


] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------
------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------
------------------
[INFO] Total time: 01:43 min
[INFO] Finished at: 2019-05-09T10:18:45Z
[INFO] Final Memory: 50M/411M
[INFO] ------------------------------------------------------
------------------
```

```






INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------
------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------
------------------
[INFO] Total time: 01:40 min
[INFO] Finished at: 2019-05-09T10:16:29Z
[INFO] Final Memory: 43M/337M
[INFO] ------------------------------------------------------
------------------
```

Fig. 12 Log comparison regression-2 of webapp-1

- A description of the regression.
- A description of the tags that leads to the regression commit $C_{regression}$ and the commit that fixes it C_{fix}.
- A description of how to run the tests on such commits.
- A description of artifacts that were generated using ElasTest, like logs, log comparisons and videos.

4.4.3 Logs

Within the dataset, we provide logs of the test (or tests) that detect the regression (JUnit logs), the logs of the backend, and the logs of the frontend. We ran the test both in the commit that introduced the bug and the commit where the bug is fixed, and logs generated in both cases have been stored and provided as part of this dataset. Therefore, researchers can use them without having to run the tests themselves.

All logs of the same type (backend, frontend, JUnit) from each of the applications look the same. The logs collected by each of them and the way they are displayed in the text file are shown below:

- **Backend logs:** These logs are collected from the backend. They contain the date and time, the Java class that launched the message and the message itself. The shape of each log line is as follows: *DATE : HOUR : CLASS : MESSAGE*.
- **Frontend logs:** These logs are collected from the web browser when the test is running. They include the date and time, the log level (DEBUG, INFO, SEVERE) [12], the component logs and the message itself. The shape of each log line is as follows: *DATE:HOUR : [LEVEL] COMPONENT MESSAGE*.
- **JUnit logs:** These logs are collected from the test execution. They contain the date and time, the log level (INFO, ERROR) [13], the component that logs the message and the message itself. The shape of each log line is as follows: *DATE:HOUR : [LEVEL] COMPONENT MESSAGE*.

[12] DEBUG: debugging messages, INFO: information messages, SEVERE: error messages

[13] INFO: information messages, ERROR: error messages

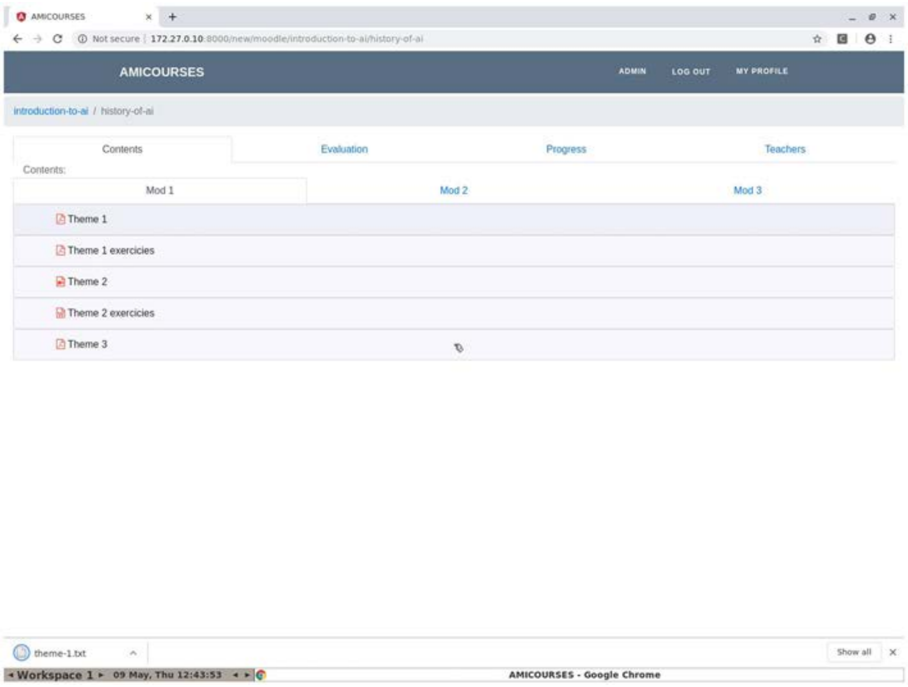


Fig. 13 Video regression-3 of webapp-2 checking if a file can be downloaded

4.4.4 Log comparison

ElasTest was used to run the tests, and it generated comparisons between the logs in C_{ERROR} and C_{BRANCH}. These comparisons might be useful to quickly detect where logs diverge, and therefore, they can provide a clue as to how behaviors are different in the version with the error, and the previous version that worked fine (See Fig. 12).

4.4.5 Videos

By using the ElasTest tool, we were also able to record the browser window. These videos were retrieved and uploaded into the dataset in the corresponding regression folder. These videos can be used by researchers to develop new techniques enabling automatic detection of UI faults (See Fig. 13). The analysis of UI regressions is a common topic in the literature Sun et al. (2020) and the industry, where tools like Applitools[14] provide artificial intelligence for visual recognition.

4.4.6 Docker image

For researchers to be able to use the dataset successfully in their research, it is of utterly importance the reproducibility of the results provided by a dataset. In this case, a special

[14] https://applitools.com/

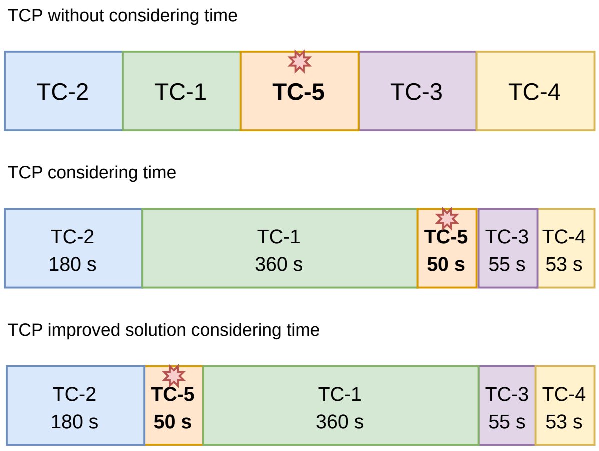


Fig. 14 Test case prioritization example. Notice how a given prioritization (top) might not be so good when time is taken into account (middle). At the bottom, a possible prioritization that mixes running time and other criteria

attention has been paid to enable this reproducibility, by different means. First, C_{BRANCH}, C_{ERROR}, and C_{FIX} are properly documented for each regression, and tagged appropriately for a quick checkout. Second, all the results were collected and ways of reproducing documented. Finally, a Docker image with all the necessary software in the versions we used, is provided, so that tests can be run and artifacts collected using the same set of tools and with the same versions.

4.5 Applications for a dataset of end-to-end bugs in software research

In this section, we will present in which cases the bugs included in our dataset may be useful for researching different problems.

Software Testing Education is one of the main lines of research where our dataset would be useful. The different artifacts offered by the dataset (logs, comparative logs or videos), the simple way to reproduce the cases where the run of the test passes successfully or not and all the documentation about each of the bugs can be used to learn more about this kind of tests (e2e) and the bugs they can reveal. Another interesting experiment could be to offer the students the specific version of the code where the tests fail, offer them the assets mentioned above and check if they are able to detect the nature of the failure in order to fix the application.

This dataset could also be used to compare the different proposals of *test case prioritization (TCP)*, a subject studied extensively in the literature Catal and Mishra (2013); Hao et al. (2016); Yoo and Harman (2012). These proposals usually use as subjects simple programs/applications, prioritizing unit tests, with low running times. For example, in FAST Miranda et al. (2018), authors use Defects4J dataset, which only contains unit test with a running time of less than a second. A recent review of the literature on TCP techniques in web services reveals that running time is not considered when prioritizing. Other metrics such as memory or CPU are not mentioned either. Therefore, time has been ignored in the literature, despite being the main reason why tests are prioritized: due to a limited time budget.

To illustrate the limitations of using unit tests in this TCP problem, we will use as an example the FAST Miranda et al. (2018) algorithm, a solution to TCP based on test code similarity. In this work, the authors use as a test subject, the Defects4J dataset, which contains only unit tests. Their solutions (the prioritized list of test cases) seek to place the most dissimilar tests first in order to try to find the faults in the system by running the least number of test cases. Figure 14 shows a prioritization that FAST could perform. Assuming Test Case 5 is the one revealing bug, finding this bug would require running first another two test cases.

At the top of the figure, we show the usual prioritization where all tests running times are considered to be the same: the unit tests of the dataset used have such a low running time that this metric is not considered when prioritizing (*TCP without considering time*). If these were e2e tests, their running times might well be very different. This is represented in the middle (*TCP considering time*) by adjusting the width of the box such that it is proportional to the running time for the test case. Notice how TC-2 and TC-1 run for much longer than the other three test cases. Could one have taken time into account to improve this solution? We hypothesize that if researchers used datasets with e2e tests, they could propose multi-objective solutions, where both similarity and time are prioritized (*TCP improved solution considering time*). Notice how by adopting a multi-criteria approach, better, compromised solutions could be found.

Adapting these proposals to the specificities of e2e tests would extend their applicability to industrial projects, where running a test before another could save minutes, if not hours.

Automatic Repair is another subject dealt with in the literature Le Goues et al. (2012); Qi et al. (2015); DeMarco et al. (2014); Xuan et al. (2015); Durieux et al. (2015) that requires a collection of documented bugs in order to build different proposals. Researchers in this field often use simple program examples. Our dataset provides repairable bugs that present a challenge, since the error can reveal in any component of the application, including in the frontend in the browser.

Bug localization is another research field that can be explored through the glasses of e2e tests using this dataset. It could be used to measure the effect of having more or less information in the search of bugs Dao et al. (2017); Zhou et al. (2012).

Bug classification is a possible research area that can be explored with our dataset, using all the data sources it offers Neelofar et al. (2012); Pingclasai et al. (2013). In applications with different components connected to the network or that make use of a database system, they can lead to classifications not previously considered. An example could be the classification according to where the bug is located (on the web client side or on the server side) or errors due to concurrent database access (when several clients access the database at the same time).

5 Comparative study of resource consumption of e2e and unit tests

The objective of this comparative study is to assess whether there are differences between the state-of-the-art datasets and the dataset proposed in this paper that justifies the creation of a dataset of e2e tests. This comparison aims to check if the differences between e2e tests (provided by our dataset) and unit tests (provided in other datasets) are significant in terms of resource usage (CPU and memory) and running time, which are critical when running these e2e tests in CI environments. Running time might render solutions of algorithms from the state of the art from problems like TCP and TSP as bad ones, by selecting or prioritizing first test cases that take long-running times, and therefore delaying other test cases that might run faster and reveal faults. Additionally, memory might have an impact as this resource is limited in CI environments. Selection or prioritization of tests could take into account memory in order to consider parallelizing the test case execution. Finally, CPU, although it is not bounded as memory and can be shared, might make test cases to fail due to timeouts. Again, taking into account CPU metrics could bring more robust solutions for selecting or prioritizing e2e test cases. These are only some examples on how these metrics could be leveraged in a couple of problems, but in other problems, it might be interesting to consider them as well when dealing with these kinds of tests.

In our comparative study, three different datasets will be taken into account: Defects4J (D4J), BugsJS (BJS) and the one presented in this work (E2E), choosing a subset of five different projects from D4J and BJS. We decided to use test cases from different projects to avoid bias due to specificities of one project. Some test cases have been selected for each dataset on which these metrics will be measured. The framework for comparison is as follows: for the dataset presented in this paper, all the three projects with all their test cases are selected. For the Defects4J and BugsJS datasets, fifteen test cases were randomly selected by choosing three random tests from each of the five projects randomly selected. Random choice was made to avoid any bias. We hypothesize that unitary tests are rather fast to run; therefore, we expect little differences between choosing one or another. Therefore, we ensured test cases from different projects were selected, but allowed random choice of test case within a project as we do not have any indication of whether a test case is more representative than another one.

In order to measure running time, CPU utilization and memory, all selected test cases were run on the same machine, an Intel i7 3.2GhZ (with eight cores) and 16Gb RAM, one at a time. Each test case is run ten times, upper and lower values are removed, and the median of the remaining ones was reported. The experiment was carried out using ElasTest, a tool that helps in collecting these metrics automatically for test cases. Through this tool, it is possible to record the logs of all test cases. In end-to-end tests, as those of our dataset, ElasTest records as well logs and metrics for all the components (web server, database and browsers), and videos for the browser interactions that tests perform. For replication purposes, a detailed description of the experiment is provided in our reproduction package [15], including ElasTest TJob (Test Job) specifications to run all the tests.

The results of this comparative study are presented in Table 3. The column dataset-project refers to the dataset to which the test case pertain (using the abbreviations mentioned above) and the specific project it belongs to. The column test case refers to the executed test case. The total time column reports the median of the total running time (in seconds) for all the processes, including downloading dependencies and compiling, that are necessary before actually running the tests. Test time reports exclusively the test running time. The AvgCPU column contains the average CPU utilization (in percentage, where 100% means a CPU is being used 100% of the time of 1 core) across all the test running time. Finally, MaxMem reports the average memory consumption (in Mb) for the test running time.

Notice that in the case of the dataset provided in this paper, tests usually involve several processes, namely a browser, a server and the testing framework itself. In order to measure running time, the time has been taken from when the first process starts until the last one ends (this process is in all cases the test framework). Regarding the average CPU utilization, the CPU of each process were measured separately, and then the sum of the averages of each process is calculated. As for memory utilization, the maximum memory used by each process was measured and then aggregated.

According to these results, there are important differences between the datasets. Unit tests in Defects4J and BugsJS take all below 1 second to run, whereas in our dataset end-to-end tests are all two or three orders of magnitude above them, ranging from 26 to 85

[15] https://github.com/e2e-tests-dataset/comparative-experiment

Table 3 Metrics gathered for the selected tests of each dataset

Dataset-Project	Test case	Total Time (s)	Test Time (s)	Avg CPU (%)	MaxMem (Mb)
BJS-Bower	testGetSource	31.455	0.002	114.466	278.708
BJS-Bower	testGetTarget	22.344	0.002	123.1	318.308
BJS-Bower	testHasNew	22.62	0.013	120.932	315.122
BJS-Eslint	testRules	64.858	0.002	111.813	957.609
BJS-Eslint	testExecutedOnText	131.909	0.383	61.909	819.397
BJS-Eslint	testIgnorePaths	65.706	0.005	109.034	960.766
BJS-Express	testAppRequest	25.736	0.035	110.746	193.06
BJS-Express	testAppResponse	16.714	0.042	116.273	212.748
BJS-Express	testAppUse	16.479	0.033	120.449	218.338
BJS-Pencilblue	testGetIdWhere	39.762	0.003	109.885	510.054
BJS-Pencilblue	testGetBodyParsers	48.732	0.002	106.99	456.028
BJS-Pencilblue	testGetEmbedUrl	39.608	0.017	108.769	525.854
BJS-Shields	testNodeifySync	47.537	0.001	118.16	662.162
BJS-Shields	testTextMesurer	57.727	0.064	121.549	574.825
BJS-Shields	testLRUCache	46.26	0.000	125.29	660.115
D4J-Chart	testGenerateURL	16.404	0.052	265.998	564.032
D4J-Chart	testCloning	16.426	0.053	266.166	569.639
D4J-Chart	testSerialization	24.633	0.079	235.297	548.636
D4J-Closure	testGetFunctionFor...	29.479	0.343	302.734	1062.073
D4J-Closure	testMergeOverflow...	20.841	0.046	334.589	1031.902
D4J-Closure	testRegExp	21.441	0.431	333.046	1081.007
D4J-Lang	test_getEnum	39.055	0.004	137.874	648.226
D4J-Lang	testLang328	48.2	0.031	130.656	646.4
D4J-Lang	testIsAlpha	39.383	0.005	139.807	644.345
D4J-Math	testInterval	59.444	0.034	153.397	951.011
D4J-Math	testCluster	50.044	0.141	151.373	928.505
D4J-Math	testLogGamma...	49.922	0.111	156.083	917.46
D4J-Time	testBasicComps1	31.958	0.014	198.817	546.464
D4J-Time	testGetMethods	31.99	0.011	200.068	543.315
D4J-Time	testIsContiguous_RP	42.276	0.006	185.523	548.605
E2E-WebApp-1	checkCreateList	184.61	27.727	129.082	2555.792
E2E-WebApp-2	checkShowProfile	152.933	26.458	66.956	2681.416
E2E-WebApp-2	checkCreateCourse	172.236	34.194	132.954	2792.77
E2E-WebApp-2	checkDownload	170.486	36.126	132.866	2823.667
E2E-WebApp-3	checkShowAdminPage	210.01	85.563	142.915	2398.944

seconds. When compared with the longest test case from BugJS and Defects4J combined (namely, testRegExp from Defects4J), that takes 0.431 seconds, the longest end-to-end test case took 85.563 seconds, meaning two orders of magnitude difference. Nevertheless, we run a statistical test to formally find if there are signification differences in running times between unit and e2e tests. Given that we have unpaired data (both samples differ in size), we used the Mann–Whitney U test (also known as Wilcoxon sum rank test). The

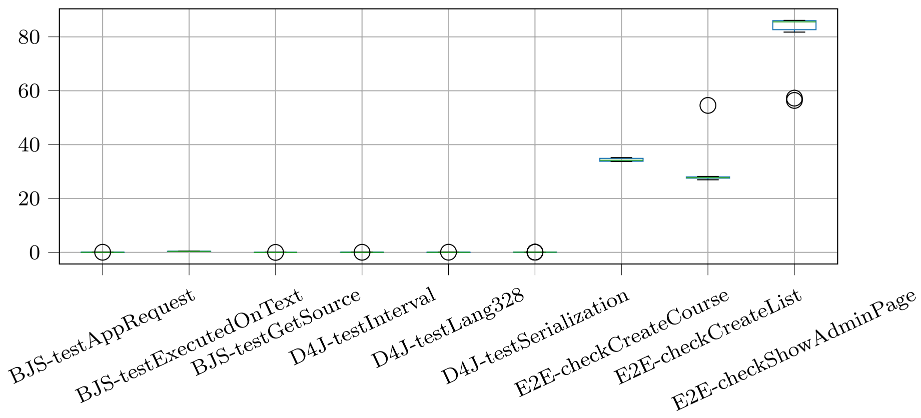


Fig. 15 Comparing test time across all test

authors established a significance level of 0.01 for the null hypothesis (both samples do not have significant differences). When running the statistical test, the authors obtained a p-value of 0.00044, therefore rejecting the null hypothesis ($p < 0.1$). Given the small sample of our dataset, the z-score needs to be taken into account. The z-score value obtained is $-3.51196 < -1.96$ thus confirming our hypothesis that unit and e2e tests differ in terms of running time. We performed a similar comparison in terms of CPU and memory. For CPU, the p-value was 0.68916, and therefore the null hypothesis cannot be rejected. In terms of CPU, there are no significant differences between unit and e2e tests. Regarding memory, the test reports a p-value of 0.00044, and a z-score of -3.51196, confirming that there are significant difference between unit and e2e tests in terms of memory consumption.

Given that authors prepared the test cases and used best practices in doing so, running times are due to the intrinsic nature of e2e tests. The test needs to start a browser, load pages, interact with the elements in the page, and wait until the results of those interactions appear in the page. In the meanwhile, the application, as a result of the interactions, is requesting data from the database, and rendering content into HTML documents that are returned to the

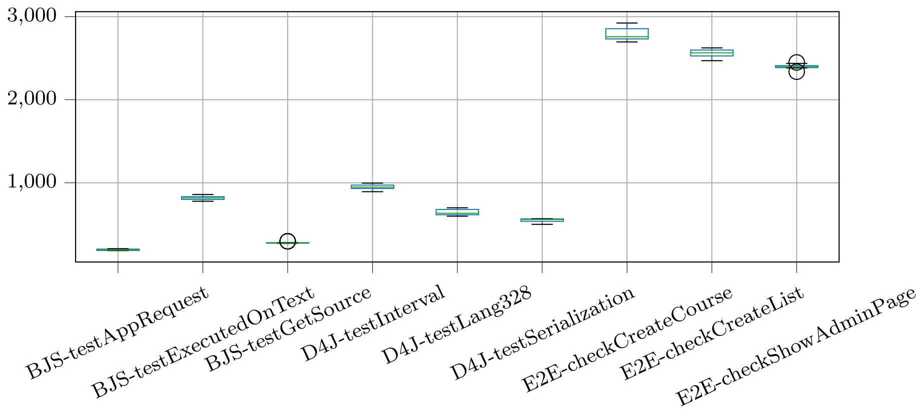


Fig. 16 Comparing max memory across all test

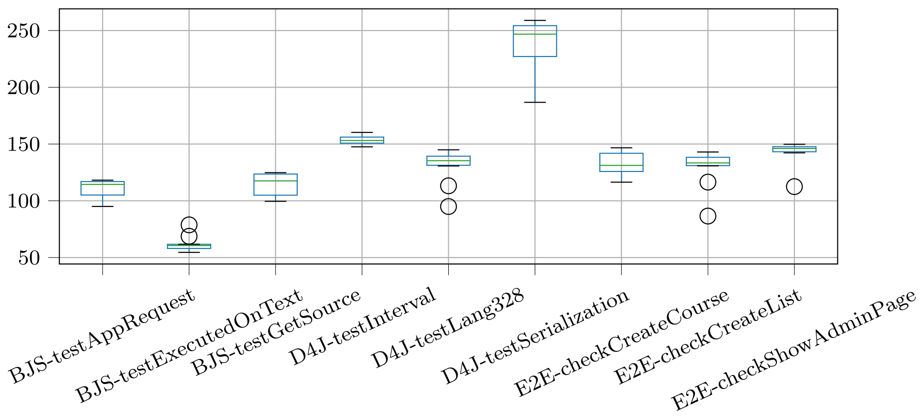


Fig. 17 Comparing average use of CPU across all test

browser. If several of these interactions happen as part of the test case, the page loading times start to sum up. As a result, running times are much longer than those of unit tests.

Regarding memory utilization, unit tests use around 500Mb, and much of that is memory used by the Java Virtual Machine and the testing framework, as they run in the same process. However, end-to-end tests in our dataset sometimes require 2.5Gb of memory. This is three times more than unit tests, and it is due the multiple process involved (browser, database and server), each one consuming a considerable amount of memory. This might be a limitation when solving some research problems related to testing, as failures might be caused by just lack of memory, something that hardly happens in unit tests.

Finally, CPU usage values are quite similar, as none of them are CPU intensive processes. It might be expected that end-to-end tests would require more CPU on average, due to the run of several processes. However, many times CPU is idle waiting for network messages or writing to disk, as in the case of the database. Therefore, in general, the CPU usage of the different processes involved is not so high, being quite similar as the CPU usage of unit tests.

In order to study the variability among different runs of the same test cases, boxplots of a subset of the test cases are shown below. Specifically, first three test cases of each dataset were selected for comparing Test Time (Fig. 15), Max. Memory (Fig. 16) and average use of CPU (Fig. 17).

Both time and memory metrics have little variability in their values for different test runs, except for the end-to-end test *checkCreateCourse* that has a higher variability. This variability might have an impact on continuous integration environments if they are not sized properly, as the test might exhaust the memory of the environment causing the test to fail. These failures might slow down a team that has to check constantly if a failing test corresponds to an actual bug or an infrastructure problem. Compared to memory and time, the variability in the values of the CPU usage is slightly higher. Although this is not as critical as in the case of memory, because CPU can be shared across multiple processes, it might cause some applications to run slowly, and timeouts for network calls might start to appear, making the test fail without really revealing any problem.

6 Limitations and threats to validity

Authors see mainly two threats to validity for the dataset described in the paper. One threat we face is related to external validity, specifically to the representativeness of the selected subjects. These projects arise from the academic field, they are not real applications. We tried to implement the functionalities of the applications they pretend to imitate without being under the property of any company, making them perfect candidates to investigate. Most of the open-source code on the internet is code libraries, which do not require e2e testing, so it is difficult to find candidate subjects to extend this dataset. The number of regression bugs is limited due to the time it takes to develop them (in the context of other changes in the project) and document them properly, but the efforts made to introduce them are similar to real industrial projects.

The other threat comes from the limitation in the number of projects and bugs. Authors aim is to increase the number of projects and bugs, both manually and automatically, so that a dataset of significant size can be provided. Nevertheless, the dataset can be extended through contributions from other researchers in order to build a bigger dataset of e2e tests.

Another threat is that in e2e testing, many smaller bugs may be hidden behind a revealed (inadvertent) bug. Due to the way the dataset has been built, with carefully defining each regression and introducing it into the corresponding feature branch, we do not think this threat may happen.

7 Conclusions and future work

This paper presents a dataset of Java applications, with their respective tests, into which regressions have been carefully introduced, in such a way that each regression is detected by an e2e test. The dataset includes not only the source code of the applications and the tests, but also extensive documentation of each of the bugs, logs for the different pieces composing the applications, their comparison and video recordings of the execution of the tests. We discussed why research works based on unit tests might not generalize well when applied to e2e tests. A comparative study with two popular research datasets containing unit tests has been conducted, showing that e2e tests are quite different from unit tests in terms of running time and memory consumption, thus deserving some research in their own. The study also revealed that in general CPU usage of e2e tests is not very different from unit tests. This work intends to be a starting point to create a dataset that allows researchers to work with a kind of tests (e2e) not usually considered in academia.

Acknowledgements This work has been supported by the Government of Spain through project "BugBirth" (RTI2018-101963-B-100), the Regional Government of Madrid (CM) through project Cloud4BigData (S2013/ICE-2894) cofunded by FSE & FEDER and the European Commission through European Project H2020 822717: MICADO.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Bertolino, A., Calabró, A., De Angelis, G., Gallego, M., García, B., Gortázar, F. (2018). When the testing gets tough, the tough get elastest. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings ACM, 17–20.

Biswas, S., Islam, M.J., Huang, Y., Rajan, H. (2019). Boa meets python: a boa dataset of data science software in python language. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), 577–581.

Catal, C., & Mishra, D. (2013). Test case prioritization: a systematic mapping study. *Software Quality Journal, 21*(3), 445–478.

Dallmeier, V., Zimmermann, T. (2007). Extraction of bug localization benchmarks from history. In: Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering, ASE '07. ACM, New York, NY, USA 433–436. http://doi.acm.org/10.1145/1321631.1321702

Dao, T., Zhang, L., Meng, N. (2017). How does execution information help with information-retrieval based bug localization? In: 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), 241–250. https://doi.org/10.1109/ICPC.2017.29

DeMarco, F., Xuan, J., Le Berre, D., Monperrus, M. (2014). Automatic repair of buggy if conditions and missing preconditions with smt. In: Proceedings of the 6th international workshop on constraints in software testing, verification, and analysis, 30–39.

Do, H., Elbaum, S., Rothermel, G. (2005). Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering, 10*(4), 405–435.

Durieux, T., Martinez, M., Monperrus, M., Sommerard, R., Xuan, J. (2015). Automatic repair of real bugs: An experience report on the defects4j dataset.

Gortázar, F., Gallego, M., Donato, M., Pages, E., Edmonds, A., Tuñón, G., Bertolino, A., De Angelis, G., Cervantes, A., Bohnert, T.M., et al. (2018). The elastest platform: Supporting automation of end-to-end testing of large complex applications.

Gortazár, F., Gallego, M., García, B., Carella, G.A., Pauls, M., Gheorghe-Pop, I.D. (2017). Elastestan open source project for testing distributed applications with failure injection. In: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 1–2.

Gyimesi, P., Vancsics, B., Stocco, A., Mazinanian, D., Árpád Beszédes, Ferenc, R., Mesbah, A. (2019). BugsJS: A benchmark of javascript bugs. In: Proceedings of 12th IEEE International Conference on Software Testing, Verification and Validation (ICST).

Hao, D., Zhang, L., & Mei, H. (2016). Test-case prioritization: achievements and challenges. *Frontiers of Computer Science, 10*(5), 769–777.

Hutchins, M., Foster, H., Goradia, T., Ostrand, T. (1994). Experiments on the effectiveness of dataflow-and control-flow-based test adequacy criteria. In: Proceedings of 16th International conference on Software engineering, 191–200.

Just, R., Jalali, D., Ernst, M.D. (2014). Defects4j: A database of existing faults to enable controlled testing studies for java programs. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014, pp. 437–440. ACM, New York, NY, USA. http://doi.acm.org/10.1145/2610384.2628055

Le Goues, C., Nguyen, T., Forrest, S., & Weimer, W. (2012). Genprog: A generic method for automatic software repair. *IEEE Transactions on Software Engineering, 38*(1), 54–72. https://doi.org/10.1109/TSE.2011.104

Liang, J., Elbaum, S., Rothermel, G. (2018) Redefining prioritization: Continuous prioritization for continuous integration. ICSE '18. Association for Computing Machinery, New York, NY, USA, 688-698. https://doi.org/10.1145/3180155.3180213

Memon, A., Gao, Z., Nguyen, B., Dhanda, S., Nickell, E., Siemborski, R., Micco, J. (2017). Taming google-scale continuous testing. In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP '17, IEEE Press, Piscataway, NJ, USA, 233–242. https://doi.org/10.1109/ICSE-SEIP.2017.16

Miranda, B., Cruciani, E., Verdecchia, R., Bertolino, A. (2018). Fast approaches to scalable similarity-based test case prioritization. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), 222–232. https://doi.org/10.1145/3180155.3180210

Myers, E. W. (1986). Ano (nd) difference algorithm and its variations. *Algorithmica, 1*(1–4), 251–266.
Neelofar, Javed, M.Y., Mohsin, H. (2012). An automated approach for software bug classification. In: 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, 414–419. https://doi.org/10.1109/CISIS.2012.132
Nir, D., Tyszberowicz, S., Yehudai, A. (2007). Locating regression bugs. In: Haifa Verification Conference. Springer, 218–234.
Pingclasai, N., Hata, H., Matsumoto, K. (2013). Classifying bug reports to bugs and other requests using topic modeling. In: 2013 20th Asia-Pacific Software Engineering Conference (APSEC), 2, 13–18. https://doi.org/10.1109/APSEC.2013.105
Qi, Z., Long, F., Achour, S., Rinard, M. (2015). An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis, 24–36.
Saha, R., Lyu, Y., Lam, W., Yoshida, H., Prasad, M. (2018). Bugs. jar: a large-scale, diverse dataset of real-world java bugs. In: 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), 10–13.
Sarro, F. (2018). Predictive analytics for software testing. In: 2018 IEEE/ACM 11th International Workshop on Search-Based Software Testing (SBST), 1.
Soto-Sánchez, O., Maes-Bermejo, M., Gallego, M., Gortázar, F. (2020). A dataset of regressions in web applications detected by end to end tests. In: R.d.S.A. Shepperd M. Brito e Abreu F., P.C. R. (eds.) Quality of Information and Communications Technology (QUATIC 2020). Springer, 1266.
Spacco, J., Strecker, J., Hovemeyer, D., Pugh, W. (2005). Software repository mining with marmoset: An automated programming project snapshot and testing system. In: ACM SIGSOFT Software Engineering Notes, 30, 1–5.
Sun, X., Li, T., Xu, J. (2020). Ui components recognition system based on image understanding. In: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 65–71. https://doi.org/10.1109/QRS-C51114.2020.00022
Widyasari, R., Sim, S.Q., Lok, C., Qi, H., Phan, J., Tay, Q., Tan, C., Wee, F., Tan, J.E., Yieh, Y., Goh, B., Thung, F., Kang, H.J., Hoang, T., Lo, D., Ouh, E.L. (2020). Bugsinpy: A database of existing bugs in python programs to enable controlled testing and debugging studies. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020, p. 1556-1560. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3368089.3417943
Xuan, J., Matias, M., DeMarco, F., Sebastian, L., Thomas, D., Le Berre, D., Monperrus, M. (2015). Nopol: Automatic repair of conditional statement bugs in large-scale object-oriented programs. IEEE Transactions on Software Engineering.
Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability, 22*(2), 67–120.
Zhou, J., Zhang, H., Lo, D. (2012). Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: 2012 34th International Conference on Software Engineering (ICSE), 14–24. https://doi.org/10.1109/ICSE.2012.6227210

Óscar Soto-Sánchez is a predoctoral researcher in the Computing Science Department at University Rey Juan Carlos, Móstoles, Madrid, Spain.

Michel Maes-Bermejo is a predoctoral researcher in the Computing Science Department at University Rey Juan Carlos, Móstoles, Madrid, Spain.

Micael Gallego Carillo is a full professor in the Computing Science Department at University Rey Juan Carlos, Móstoles, Madrid, Spain.is a full professor in the Computing Science Department at University Rey Juan Carlos, Móstoles, Madrid, Spain.

Francisco Gortázar Bellas is a full professor in the Computing Science Department at University Rey Juan Carlos, Móstoles, Madrid, Spain.