Enhancing the Learning of Database Access Programming using Continuous Integration and Aspect Oriented Programming*

Beatriz Pérez
Departamento de Matemáticas y Computación
Universidad de La Rioja
Logroño, Spain
beatriz.perez@unirioja.es

***Abstract*—Database access programming is a noteworthy component of Software Engineering (SE) education on databases that students are expected to acquire during training for their careers. In our university, we cover such an education in a course that emphasizes the use of the JDBC API to access databases. This paper presents our experiences in developing and running a framework to enhance the learning experience of database access programming, which is motivated by several factors. First, our students face great demands on acquiring JDBC acknowledge, and providing them with constructive feedback serves a critical role. Second, the increasing number of students leads to high efforts in managing and grading their assignments. Finally, we consider of strategic importance to bring modern industrial SE techniques into the classroom, so that students obtain a better experience with industry practices.**

Our framework draws upon constructive alignment and automated formative assessment, combining Continuous Integration (CI) and Aspect Oriented Programming (AOP). We include an innovative application of AOP, a programming technique that aims to modularize inherently scattered functionality into single functional units, to help students adopt well-established JDBC best practices. We also use well-known industrial software tools (*Travis CI* and *GitHub*) to manage and grade students' assignments and support automated integration testing with databases. The findings of this study, applied to a class of 53 students, suggest positive effects, such as motivate students to implement JDBC best practices, streamline the management and grading of their assignments, help them get familiar with industrial tools, or improve their grades.

***Index Terms*—JDBC, Travis CI, GitHub, AOP, Integration tests, Best practices**

I. Introduction

Software engineering (SE) methods and practices, used in industrial development projects, are in continuous change. With this in mind, it is of strategic importance that SE education is responsive for such changing industry trends, aiming at bridging the gap between current SE education and software industry [1]. In the last decades, there have been different attempts from universities and other higher education institutions to bring modern industrial software development techniques into the classroom [1]. Among such industry practices, we note version control, code reviews, or Continuous Integration (CI) [2] which mainly aims at automating the integration of code changes from multiple contributors into a single software project located in a central repository, after which automated builds and tests are run. These practices are supported by a range of techniques, tools and technologies, such as *Travis* for CI, or *GitHub* for version control [2].

In the particular case of programming courses, several proposals have attempted to replicate modern industry practices in the classroom. Most of those works use CI tools to teach basic programming courses, normally, in the first years of SE degree programs at universities [3]–[5]. In these courses students learn basic concepts of programming languages (such as Java or C++) and practice, among others, design and implementation of code in a CI environment. Later, students' deliveries are checked against *unit tests* where students' classes and methods are tested. The skills and acknowledge acquired through first programming courses make students qualified to face the challenges implied by other programming aspects that encompass an added difficulty. This is the case of database access programming, which is a noteworthy component of SE education on databases (DBs). In the university where this work has been conducted, we are covering such an education in a course in the second year of a SE degree program. The course addresses the use of Java Database Connectivity (JDBC) [6], which is a mechanism that allows Java to communicate with relational databases. As part of the course, students have to perform several programming exercises that use JDBC to access DBs. In this case, instead of unit tests, students' programs are checked against *integration tests*, where the combination of such programs with DBs is tested. Both designing and managing integration test entails an added difficulty because of the need to consider additional components (databases) to combine [7]. Following software industry's recommendations, teachers also encourage students to adopt (the often forgotten) well-established JDBC best practices, promoting better software quality and sustainability. Until now, students submit their assignments through the course's management system (CMS) and teachers have to check them manually on their computers, which makes of managing and grading deliveries a hard task (Section II).

In this paper, we outline our proposal for overcoming such challenges, which keeps in mind six main motivating ques-

tions, distinguishing among those student-focused: 1) motivate and encourage students to implement JDBC best practices, 2) improve the development and delivery of their assignments, easing automated testing, 3) help students get familiar with modern industrial tools, and 4) improve their grades; and those teacher-focused: 5) streamline the management process of students' assignments, and 6) their grading. To achieve such goals, we have set out a framework that is drawn upon constructive alignment and automated formative assessment, combining (i) the integration of CI and version control (*Travis CI* and *GitHub*) in our course, to manage and grade students' assignments and to support the execution of database integration tests, and (ii) a novel application of the Aspect Oriented Programming (AOP) paradigm [8] which, together with our integration tests, implements our automated formative assessment approach to help students reach JDBC learning goals, focusing mainly on the adoption of best practices (Section III). We have applied our framework to 53 students attending the course. The findings drawn from the conducted case study show that the six main motivated intentions are achieved (Section IV). Related work and a discussion about these findings, including limitations, challenges and lessons learned, appear in Sections V and VI, respectively. Finally, conclusions are drawn in Section VII.

II. Course Background and Motivation

Our programming course includes in its program principles and concepts involved in the development of database access applications, such as 1) general issues of architectures of database applications (focusing on the three-tier architecture: *presentation*, *business* and *persistence* layers), and 2) techniques to implement the *persistence* layer (such as JDBC or Object-Relational Mapping -ORM- frameworks). JDBC constitutes the main part of the course syllabus, where students are instructed about basic and advanced aspects of this library. Traditionally, the format of the course alternates master classes (28 hours in total) with 11 laboratory sessions (2 hours each), to transmit conceptual knowledge and procedural skills combining theory and practice. We devote 6 master class hours to teach JDBC, where lectures are also interspersed with guided group discussion periods and problem solving activities. Students perform 5 JDBC laboratory assignments (one session per each assignment), where they are asked to develop JDBC programs which interact with different MySQL/Oracle DBs. These activities allow them to consolidate the knowledge transmitted in master classes. Besides the JDBC library itself, students learn JDBC best practices (BPs) as suggested in specialised bibliography on the topic [9], [10], which mainly help write effective and robust JDBC code, while ensuring optimal performance of it. The main intended learning outcome of the course regarding JDBC is that the students acquire the knowledge and skills necessary to properly use this library, paying special attention to the adoption of well-established JDBC BPs. For a better understanding of our approach, we next describe some JDBC background information, and go on to present the issues that motivate this work.

```
1 Connection conn = null;
2 PreparedStatement stmt1, stmt2, stmt3 = null; ResultSet rs = null;
3 try {
4    conn = DriverManager.getConnection(<parameters>);
5    conn.setAutoCommit(false); //Turn off the auto-commit mode
6    stmt1= conn.prepareStatement(SelectSQL);//Create PreparedStatement
     ...
7    rs= stmt1.executeQuery();//Execute the Select and get the ResultSet
8    while(rs.next()){ //Iterate through the ResultSet and manage results
9       Integer id =rs.getInt(1);//Use getXXXs referencing columns by index
10      if( rs.wasNull() ) {  id = null; } { //Check null values
        ...
11   }
12   stmt2=conn.prepareStatement(ModificationSQL1);//Create PreparedStatement
13   stmt2.executeUpdate(); //Execute the Modification instruction
     ...
14   conn.commit(); //Commit transaction
15 }
16 catch (SQLException e) {
17      try { if (conn != null) conn.rollback();} //Rollback transaction
18      catch (SQLException e) { //Handle }  //Handle
20 }finally { //Clean up all resources
21      try {if(rs/stmt1../conn != null ) rs/stmt1../conn.close(); }
22      catch( SQLException e ) { //Handle}
23 } //end finally
```

Fig. 1: Code fragment illustrating a usual JDBC program

A. JDBC Fundamental

The JDBC API [6] consists of a collection of (mainly) interfaces and classes which allows Java programs to interact with databases. Figure 1 illustrates a fragment of a usual JDBC program which executes both data retrieving (`Select`) and modification instructions (`Insert/Update/Delete`). Briefly speaking, to start working with JDBC, first you need the `DriverManager` class which is used to keep track of available JDBC drivers and to generate a database connection, that is, a `Connection` object which represents the interconnection between the Java client program and the database. Connections allow generating database statements and are represented by the `Connection` interface, whose main methods are `close`, `commit`, `rollback` and `setAutocommit`. These three last methods are used when enabling manual-transaction support instead of the auto-commit mode that JDBC uses by default. `setAutocommit` is used to set the auto-commit mode so that, if it is turned on, every SQL statement is committed to the database upon its completion and, if it is turned off, the SQL statements would need to be committed/rollbacked explicitly by using the `commit`/`rollback` methods, so that all statements are grouped in one transaction (see lines 5, 14 and 17). The `Statement` interface provides methods for executing queries, such as `executeQuery`, for retrieving data (`Select`), and `executeUpdate`, for modifying data (`Insert/Update/Delete`). `PreparedStatement` is a `Statement` sub-type which allows executing pre-compiled SQL statements on databases (thus, improving performance), and prevents SQL injection attacks [6]. When executing a `Select`, you get back a `ResultSet`. The `ResultSet` interface is used for accessing the set of results returned by a `Select` (see the `next` method in line 8). A `get`*`XXX`* method for each data type *`XXX`* can be used to retrieve column values (using either column index or column name as parameter, being this latter option not recommended because of performance reasons [9]). When retrieving a `null` value from a non-primitive column, the corresponding `get`*`XXX`* method gives

TABLE I: Best practices and defined AOP pointcuts

BP	Description	Pointcuts
BP1	Properly end up all resources (`ResultSets`, if any, `Statements`/`PreparedStatements` and `Connections`). *Figure 1: line 21*	Per each JDBC resource, two pointcuts are defined which check if such a resource has been created/closed, respectively, that is, these two pointcuts capture the invocation of the corresponding create/close methods, respectively (8 pointcuts).
BP2	Use `PreparedStatements` when possible, instead of `Statements`. *Figure 1: lines 2, 6 and 12*	The pointcut defined to check if a `Statement` is created, is used to check that `Statements` are used (instead of `PreparedStatements`).
BP3	Disable *autocommit mode* when performing modifications on the database, and *commit/ rollback* when corresponds. *Figure 1: lines 5, 14 and 17*	Pointcuts are defined to check if, when the `executeUpdate` method is invoked, `setAutocommit/commit` methods are also invoked, that is, a pointcut captures de invocation of the `executeUpdate` method and two ones capture the call to `setAutocommit` and `commit`, respectively (3 pointcuts).
BP4	Reference table columns by number when retrieving row values with `getXXX`. *Figure 1: line 9*	A pointcut is defined to check if `getXXX` methods are invoked with a string as argument representing a column name, that is, such a pointcut captures calls to any `getXXX` method with a string as parameter (1 pointcut).
BP5	Use `wasNull` to check if retrieved rows lack values. *Figure 1: line 10*	Pointcuts are defined to check if, when a `getXXX` method is invoked being `XXX` a primitive type, the method `wasNull` is also invoked, that is, a pointcut captures the call to any `getXXX` method and another one captures calls to `wasNull` (2 pointcuts).

back a `null` constant, but if it is primitive, it gives back a 0 (e.g., `getInt` returns 0 if the value retrieved was either 0 or `null`). Thus, `wasNull` is used to know if the last column extracted from the `ResultSet` was `null` (see line 10). In the course we also teach a set of JDBC best practices, described in Table I, collected from specialised bibliography [9].

B. Problem Definition

As for the management of assignments, the course's CMS only allows distributing and delivering them, that is, it does not provide an integrated environment for grading them. Until now, students deliver their assignments through the CMS and then the teacher has to manually check them on her/his personal computer. Per each JDBC method developed by a student in an assignment, not only does the teacher must verify its correct functionality, but also she/he must check if the student has adopted JDBC best practices. Regarding checking functionality, until now, it is performed by verifying if each method passes its integration tests. This process gets complicated if the student's code does not pass the tests or even it does not compile, increasing the time and effort devoted by the teacher to find the problem. As for checking the adoption of best practices, it requires the instructor to go through the code line by line, informing the student about not adopted best practices. This task turns out to be a lot more work, since we have noticed that students tend to give best practices scant attention. Another aspect to consider refers to when students ask for office hours to get feedback on their work. Until now, they come to the teacher's office with their work stored in their USB flash drives, which the teacher inserts into her/his personal computer, with the security risks it may entail, also taking up valuable time.

In this process, tests play a significant role, both for teachers (for grading), and for students (for evaluating their projects during development). However, integration test design brings an added difficulty against unit tests design. For unit tests, this is easy as long as we stay away from global state. However, this becomes a bit of a challenge with integration tests that interact with DBs, since one test can corrupt the DB and cause subsequent tests to fail. Thus, it requires the DB to be in a consistent starting point each time a test is run. Until now, either students while developing their assignments or teachers when grading the deliveries, needs to manually clean the database and sets up data between test runs. This tedious task results in a disadvantage added to the just mentioned drawbacks around assignments' management and grading.

III. Overview of our framework

Aiming at facing the previous challenges, we have based upon *constructive alignment* considering also *automated formative assessment*, combining Continuous Integration (CI) and Aspect Oriented Programming (AOP). *Constructive alignment* [11] is based on the idea that teaching and learning activities, as well as assessment tasks, must be *aligned with* the outcomes students are expected to achieve from their participation in the course. Assessment tasks are often classified into *formative*, *interim* and *summative* [12]–[14]. *Formative* assessment aims to assess students' performance during the instructional period, for the purpose of providing feedback that they can use to improve their learning. *Interim* assessment, which takes place several times during the instructional period, aims both to assess students' knowledge and skills relative to a specific set of academic goals, and to inform decisions at the classroom level and beyond. *Summative* assessment, which is conducted at the end of the learning process, aims to assess how well students have performed on a certain task, typically to determine a final grade [12]–[14].

In the course at hand, the students' performance is evaluated through three evidences of learning, each mainly associated with one of the previous assessment types. First, 5% (0.5 points) of the course grade corresponds to students' efforts in the 11 *formative* laboratory assignments (each of the 5 JDBC assignments is worth around 0.045 points). Second, students take 2 *interim* partial exams: A1, taken after the two firsts JDBC assignments, and A2, taken after all the JDBC assignments, which are worth 5% and 20% (0.5 and 2 points), respectively. Third, representing the remainder 70% (7 points), a *summative* final exam is taken at the end of the course, where JDBC is not evaluated. Thus, starting from the outcome we intend students to learn regarding JDBC, and considering the teaching/learning activities defined to enable learners to meet that outcome (as presented in Section II), we have revised our formative assessment methods to ensure they are consistent with such outcome, keeping in mind our identified problems. More specifically, we have: 1) revised the testing methods included in our JDBC assignments so

as to provide students with appropriate formative feedback to enhance their JDBC learning, and 2) adopted well-known industrial software tools both to streamline the management process of students' assignments and to support the automated execution of such tests. At this point it is worth noting that, although we mainly focus on providing students with support to reach JDBC learning goals through our JDBC assignments (formative), such assessment items also help teachers assess how well students have learned (summative). Grading practices or assignments has long been a controversial issue among educators and academics, that we will discuss later on.

A. Automated Formative Assessment

The most common form of assessment for programming assignments is to check that the program functions according to the given requirements [15]. However, in our case, checking just for functionality does not ensure us that students have adopted JDBC best practices. For this reason, we need to use specific methods to check both aspects.

Regarding functionality, as stated before, the integration tests used until now lead us to face several challenges when interacting with databases. Aimed at addressing them, we have decided to use DbUnit [16], a database testing framework based on JUnit which allows us to setup and teardown a database between tests, as well as to check expected table contents once a test completes. Before each test run, DbUnit exports to the database an XML file with the dataset of the initial database state. Once the test completes, it makes comparisons between the actual data in the database, and the expected datasets (given also as XML files), so as to determine the test's success or failure. In particular, students might use test failures as hints of errors in their code.

As for the adoption of best practices, we have based on the Aspect oriented programming (AOP) paradigm [8], [17]. AOP promotes software design so that the designer focuses on the functional (*core*) concerns of a system as opposed to non-functional concerns (e.g., logging or security). Non-functional concerns tend to cut across the system rendering it difficult to understand, maintain, and modify. AOP builds on top of existing programming methodologies, augmenting them with constructs to modularize these *crosscutting concerns*, while the *core concerns* are implemented using the chosen base methodology. AOP allows a developer to modularize these crosscutting concerns into entities called *aspects*, which can then be woven into the core code by an *aspect weaver*, building the final system. Among the AOP language extensions for Java, we use AspectJ [18], which expresses crosscutting mainly through *join points*, which are well-defined points in a program execution (e.g., objects' creation or methods' call), and *pointcuts*, which are distinguished selections of join points that meet some specified criteria (e.g., the call of a method with a certain name or with a parameter of a concrete type). After a *pointcut* selects *join points*, the focus is augmenting those *join points* with additional or alternative behaviour (e.g., that related to logging). *Advices* are method-like constructs defining such a complementary crosscutting

```
1  pointcut connection_create() : withincode(* *.Persistence.*(..)) &&
2                                 call(* java.sql.DriverManager.getConnection(..));
3  pointcut connection_close() :  withincode(* *. Persistence.*(..)) &&
4                                 call(* java.sql.Connection.close());
5  after():connection_create(){ // Advices
       //We access the array associated to the tested JDBC method, of the dataMap map
6      int[] aux=dataMap.get(thisEnclosingJoinPointStaticPart.getSignature().getName());
7      aux[6] += 1; //aux[6] stores the number of created Connections
       // We update the array in the map
8      dataMap.put(thisEnclosingJoinPointStaticPart.getSignature().getName(), aux);}
9  after():connection_close(){ …
10     int[] aux=dataMap.get(thisEnclosingJoinPointStaticPart.getSignature().getName());
11     aux[7] += 1; //aux[7] stores the number of closed Connections
12     dataMap.put(thisEnclosingJoinPointStaticPart.getSignature().getName(), aux);}
13 protected void JDBCAdoption(...) {   ...
14   if (aux[6] != 0 && aux[7] == 0) {
15      logger.log(Level.INFO, indent+ANSI_YELLOW+"\t Check your code! Your JDBC method
16      creates a connection but, after finishing it, it does not clean up the resource.");
17   }else if (aux[6] != 0 && aux[6] == aux[7]) {
18      logger.log(Level.INFO, indent+ANSI_GREEN+"\t Well done! Your JDBC method creates
19      a connection and you also clean up the resource."); ...
```

Fig. 2: Code fragment of our aspect

```
****************** Method: Employee sol.Persistence.getEmployee(String) ******************
Args: 1- AGar01
    ************* Method: Branch sol.Persistence.getBranch(Connection,String) *************
    Args: 1- ConnectionImpl 2- BrZa01
    Well done! You create as many PreparedStatements as the ones you close (1)
    Well done! You create as many ResultSets as the ones you close (1)
    Well done! Your JDBC method takes a connection from another method and you just uses it.
    Check you code! You are invoking getXXX methods using column names, which is not recommended.
    ************ Method: End Branch sol.Persistence.getBranch(Connection, String) ***********
Well done! You create as many PreparedStatements as the ones you close (1)
Well done! You create as many ResultSets as the ones you close (1)
Well done! Your JDBC method creates a connection and you also clean up the resource.
Check you code! You invoke getXXX methods with primitive types, but you do not check for null values.
****************** Method: End Employee sol.Persistence.getEmployee(String)******************
```

Fig. 3: Best Practices feedback report for a JDBC program

behavior at join points. Depending on the declaration, advice bodies are executed *before* or *after* a specified join point, or they can surround (*around*) a join point. Finally, *aspects* embed crosscutting logic by including the defined *pointcuts* and *advices* [18].

We have based on AOP to monitor the JDBC methods developed by each student while they are tested, so that not only are we able to check if the student adopted JDBC best practices, but we can also provide the student with feedback on her/his progress regarding the implementation of best practices. To the best of our knowledge, it is the first work that proposes this paradigm to check the adoption of programming best practices. We have designed an AspectJ *aspect*, called *AOP module* (available at [19]), which is automatically applied each time a student's JDBC method is run, monitoring it. This aspect defines suitable *pointcuts*, 15 in total, to capture occurrences during the execution of each JDBC method. The captured data allow us to identify if the method implements BPs. Table I shows, per each BP, a description of the *pointcuts* defined to check its adoption, while Figure 2 shows an extract of our aspect. For example, to check if *connection*s are ended up, we define two *pointcuts* to capture calls to `getConnnection`, in order to know if a connection has been created, and to `close`, in order to know if such a connection has been finally end up (see lines from 1 to 4 in Figure 2). We note that the aspect searches for occurrences that take place in methods within a Java class named `Persistence`, which corresponds to the name students must give to the class that implements the persistence layer in each assessment (see lines 1 and 3). The defined *pointcuts* are accompanied by suitable *advices*, each one devoted to record the number of times the associated

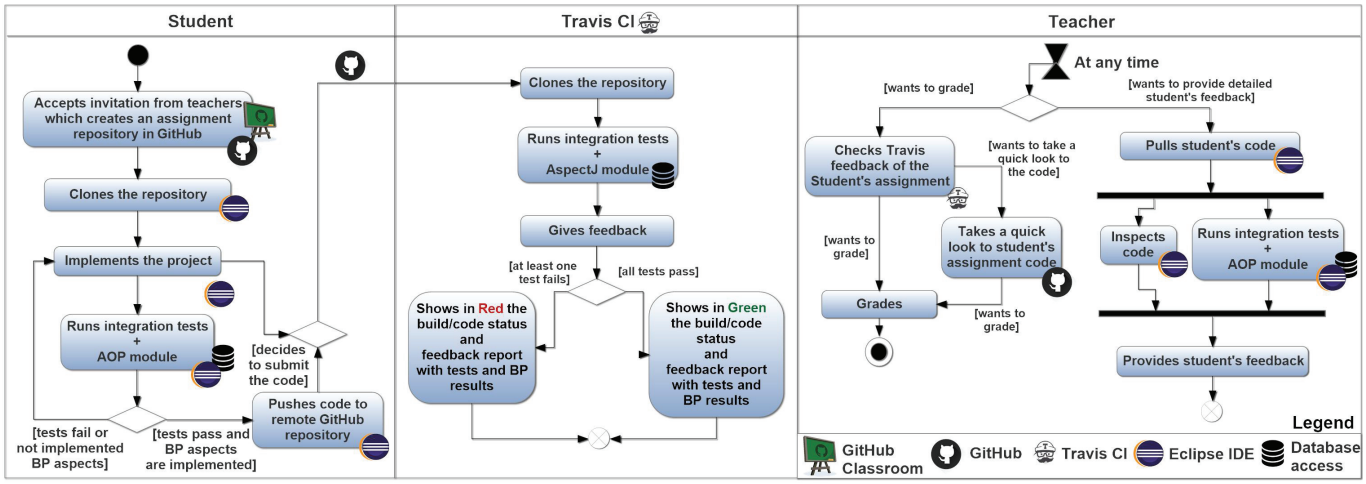

Fig. 4: Workflow per assignment

join point is matched during the execution of the JDBC method being tested (e.g., see the *advices* defined in lines from 5 to 8 and from 9 to 12, respectively). Such data is automatically stored by the *advices* in a Java Map. This Map has, the names of the tested JDBC methods, as keys, and arrays with the data gathered by the *advices* for such methods, as values (e.g., lines 6-7 and 10-11). Finally, an additional method (see lines from 13 to 19) performs specific checks per each tested JDBC method (e.g., that the number of calls to `getConnnection` and `close` match), using the data in the associated array, and logs suitable coloured messages into the console of the student's IDE (our students use Eclipse as IDE), informing about the BPs implemented by the tested method. Another pair *pointcut/advice*, respectively, captures code exceptions, and parses the overall stack trace into an only log message showing the exception class, a link to the code line, and details about what caused the exception. As a result, each test run displays, for the JDBC tested method, a coloured feedback report similar to the one in Figure 3 showing, when applicable, hints of the execution errors in the code (in red), not adopted BPs (in yellow) and motivational messages (in green), using indented text when invoking nested methods.

B. Managing Students Assignments

For managing students' assignments, we have used *Travis* as one of the most popular CI open-source services which can be easily integrated with cloud repositories, such as GitHub, an open-source repository service for collaboration and version control. Specifically, based on the steps given in [20], we have created an *organization* in GitHub Classroom which allows us to create programming assignments and distribute them to the students enrolled in the course. Then, following [21], we have set up *Travis* so that 1) it has access to our *organization* (and thus to all students' assignments), and 2) it is able to automatically run suitable tests designed by the teacher every time new code is pushed up to a student's repository. After that, per each course's assignment, we just need to set up a template repository in GitHub Classroom, including a *Travis* configuration file (with, e.g., database installation/configuration aspects, or data regarding how to setup and run tests), the instructions for students, the starter code, and a test suite (with our integration tests and the AOP module) that will run against the students' code checking for its correctness and best practices.

Figure 4 depicts a workflow with the interaction between participants and the *Travis* CI server, where each swimlane groups the activities performed by each actor. Having defined an assignment with the template repository to be completed by each student, GitHub Classroom allows instructors to send out an invitation to students, which, when accepted, will automatically setup a new private repository in GitHub for that student for the current assignment (see the *Student*'s swinline in Figure 4). The students can easily clone the repository in Eclipse to start coding and run integration tests. Each time a student runs a test, best practices are also automatically checked for compliance by using our AOP module. Both the tests and the AOP module are used as a self-evaluation tool to check which test has passed/failed and at what extend best practices have been implemented, displaying the corresponding feedback report as coloured messages. The student can work on her/his assignment either until it is completed or until the due date for the assignment. We thus help students by providing feedback on their work and let them improve it accordingly. Since students do the assessment work at least partly by themselves, it also reduces the teacher's assessment workload. At any moment during the development of the assignment and, in particular, when the student wants to hand in the assignment, she/he can decide to push the code to her/his GitHub repository to be accessible via the web (see bottom of the *Student*'s swinline in Figure 4). GitHub will automatically notify *Travis* of the change, which in turn, will clone the repository and automatically run the configured integration tests, together with the AOP module (see *Travis*'s swinline in Figure 4). Whenever students want, they can also check *Travis* feedback through the corresponding *Travis*' URL, which shows, in addition to the information displayed in the

Eclipse console, the build status in green colour, if all tests pass, or in red, otherwise. When the student asks for help (or even when the teacher wants to grade a student), the teacher can pull the student's repository from Eclipse, run the tests, inspect the code, and give her/his comments of praise, criticism or advice (see the *Teacher*'s swinline in Figure 4). Furthermore, teachers can easily check the correctness of a student's build by consulting *Travis* feedback, or access the online GitHub assignment's repository to revise the code. This way of working allows creating a real-time feedback loop between instructor and students, and streamline the management process and grading of deliveries. Supplementary material, with a template assignment and a more detailed explanation of the proposed workflow, is available at [19].

IV. Findings

Next, we highlight the results obtained from the application of our framework to a class of 53 students in the last academic year 2019/20. We based on four data sources: students' qualifications, logs obtained from the execution of the students' tests, feedback gathered from the students, and teachers' perceptions.

A. Students' qualifications

As described previously, the JDBC skills acquired by the students are evaluated through two varied evidences of learning. On the one hand, 5 JDBC laboratory assignments, which are worth around 0.045 points each. On the other hand, students have to take two partial exams: A1 and A2, which are worth 0.5 and 2 points, respectively. The remainder 7 points correspond to the final exam, where JDBC is not evaluated. JDBC laboratory assignments and exam A1 are low-stakes assessments we mainly use (as other authors argue [22]–[24]) to encourage students to be actively engaged in their JDBC learning and pay more attention to course content, while they get more prepared for high-stakes assessments (i.e., exam A2). Given the low weight of such assessment strategies, in this analysis we have decided to base on exam A2 marks. We have compared the marks obtained by the students who have attended the course in 2019/20, with those obtained by the students in the previous year 2018/19. Both courses were taught by the same instructor (the author of this paper), which provides a control for instructor variability and allows for more accurate comparisons of students' JDBC learning.

The results obtained from our analysis were very good, even better than expected. On the one hand, results show that students in 2019/20 raise their score average by 59.57% (means 1.50, in 2019/20, and 0.94, in 2018/19). On the other hand, we have asked ourselves if we could come to the conclusion that our proposal has an effect on the scores of students attending the course in this last academic year, and that there is a significant difference between the scores of both courses. Thus, the null hypothesis (H_0: $\mu_{18/19} = \mu_{19/20}$) assumes that the independent variable (learning method) has no effect on the dependent variable (score), that is, there are no differences between means, while the alternative hypothesis (H_1: $\mu_{18/19} \neq \mu_{19/20}$) assumes that the learning method has an effect on students' scores (i.e., the means are different). After performing a *t-test*, the results yielded ($t(90)=5.74$, $p<0.01$, $d=1.15$) statistically significant differences between the exam scores of both groups, so we can reject the null hypothesis and accept the alternative one. Cohen's [25] also showed a large effect (d=1.15). Furthermore, the instructor detected a significant improvement regarding the adoption of BPs in students' assignments in 2019/20. Since both groups are considered to be equal in terms of the knowledge level concerning the course, we could conclude that the method has a significant effect on the scores. To replicate this analysis, the data and a documented R script can be found at [19].

B. Logs from tests' executions

Aimed at finding students' trends regarding how they adopt best practices, we have configured our AOP module to output formatted messages not only to the Travis/Eclipse console (as described previously), but also to a log file (in a CSV format for data exploitation, including the data shown in console and also timestamps). Each student's assignment repository contains such a log file, which is populated in each test run.

We have used Process Mining Techniques for extracting information from such log files. Generally speaking, the basic idea behind process mining is to extract knowledge from event logs recorded by an information system. An event in an event log refers to an activity and is related to a particular process instance [26]. In our particular situation, the AOP module creates a new event in the external log file each time a JDBC method is tested, recording the information regarding the best practices adopted by the method. There are different process mining techniques to analyze log files. As stated in [27], dotted chart and other similar formats that organize temporal data along a time axis are easy to interpret and can allow for visual identification of patterns. More specifically, a doted chart shows the spread of events over time by plotting a dot for each event in the log. In our particular case, we have used the Dotted Chart Analysis utility of the ProM process mining framework [28] to investigate patterns of students' behaviour regarding the implementation of best practices between test runs. For such a task, per each student, we have first merged the log files obtained from her/his assignments into an only log file and, later, we have processed it obtaining a log file with 4 columns including: the anonymous student ID (*student*), the name of the tested method (*method*), the timestamp associated to the test run (*timestamp*), and the level of adoption of best practices by the method in such test run (*adoptedBPs*). This last column contains a value ranging from 0 (the method does not adopt any BP) to 4 (all BPs have been adopted). Finally, taking this file as source of the Dotted Chart Analysis utility, it spreads, per each method, its test runs over time, showing each test run in a different colour depending on the value of the *adoptedBP* column. Among the results obtained from this analysis, we have observed that students are used to test all methods each time they test each assignment, instead of testing just the method they are developing. Additionally, we

N	Questions
Q1	Do you find GitHub useful to store your assignments, so that the teacher can easily access them to provide immediate feedback?
Q2	Do you find it easy to hand in your assignments through GitHub?
Q3	Do you find it easy to use GitHub from the Eclipse IDE?
Q4	Would do you use GitHub for your personal projects?
Q5	Do you find Travis suggestions useful?
Q6	Do you feel confident when using integration tests to check the functionality of your JDBC code?
Q7	Do you consider DbUnit useful to setup databases for integration tests?
Q8	Have you found it useful to use the AOP module to check the implementation of JDBC best practices (BPs)?
Q9	Do you think that the use of the AOP approach has motivated you to pay more attention to the implementation of JDBC BPs?
Q10	Do you think it would be a good idea for other courses to use AOP to implement BPs in other programming languages?

Fig. 5: Likert scale and open questions

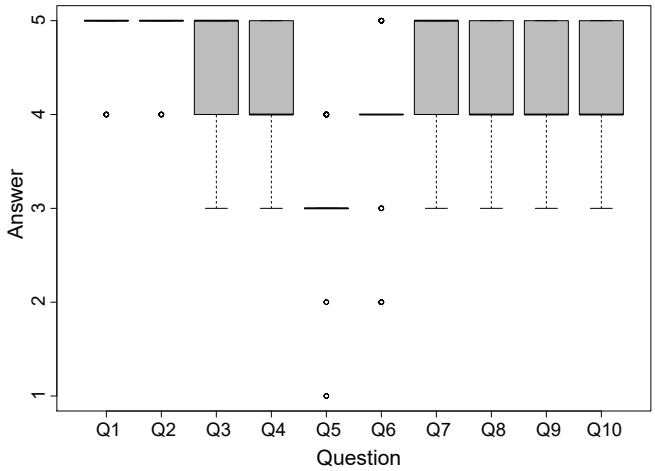

Fig. 6: Boxplot of the likert scale analysis (the ○'s are outliers).

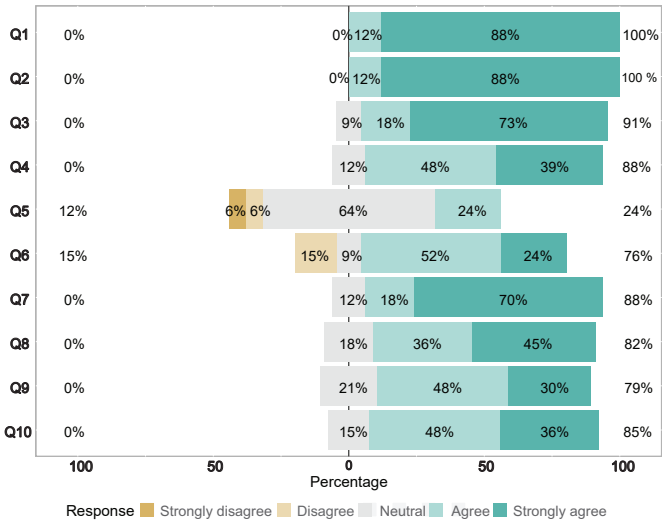

Fig. 7: Diverging stacked bar chart of the likert responses.

have found out that students usually start testing methods with not, or scarcely, implemented best practices, and finally go through the complete, or almost complete, adoption of BPs. Supplementary material, including log files and their analysis, can be seen in [19].

C. Feedback from students

To know students' perception on the experience, we conducted a voluntary and anonymous survey composed of a set of 13 questions: (i) 10 mandatory Likert scale questions (with responses from '1 - *strongly disagree*' to '5 - *strongly agree*'), where students could give their opinion on three different aspects (GitHub and Travis, integration tests and the AOP module), (ii) 1 optional open question, where students were encouraged to reflect upon the proposal, and make their comments, regarding their experiences and suggestions, respectively, and (iii) 2 questions asking students to choose between GitHub and the course's CMS to perform different tasks. 33 out of 53 students took part in the survey (62,26% response rate), perhaps because it was optional.

Figure 5 shows the Likert scale and open questions, while students' responses are shown by means of the boxplot in Figure 6, and the diverging stacked bar chart with percentages in Figure 7. The first five questions (from Q1 to Q5) refer to students' perception of the use of GitHub and Travis. The usefulness of GitHub both to store (Q1) and deliver (Q2) their assignments was rated very high (see, e.g., the very high medians in Figure 6). Most of the students find it easy to use GitHub (Q3). As for Q4, students reflect a positive appraisal of the use of GitHub for their personal projects (see, e.g., the high median in Figure 6). Considering students' comments from the open question, some students report that they feel really comfortable using GitHub for assignments' management and positively value the use of both GitHub and Travis in the course, so that they can get familiar with commonly used industrial tools. Although a small majority of students are neutral regarding if they find Travis suggestions useful (see the median or the 64% value of question Q5 in Figure 6 and Figure 7, respectively), 24% of students think they are helpful (see Figure 7). Students' assessment regarding integration tests is shown in the answers to the next pair of questions (Q6 and Q7). As for Q6, a large amount of students feel confident when using integration tests to check code's functionality of their JDBC methods (see, e.g., the high mean in Figure 6). As comments, some students positively value to get instant feedback regarding if they are on the right track. As for Q7, a representative sample of the responses consider the use of DbUnit useful to manage test data (see the very high median in Figure 6 and 88% in Figure 7). In the open question, some of them positively appreciate its independence on any particular database environment. The last block of questions, from Q8 to Q10, deals with the students' perception of the AOP proposal for checking JDBC best practices. Most of the students agree on the effectiveness of the AOP aspect to help them check the adoption of best practices (Q8). As for question Q9, results indicate that most of the students feel that the use of the AOP module motivated them to give more

attention to the implementation of JDBC BPs. When asked about whether it would be a good idea for other courses to adopt AOP to implement BPs in other programming languages (Q10), most of students agree (see, e.g., the high median in Figure 6). Some students also report that they really appreciate the feedback (hints, not adopted JDBC BPs and motivational messages when they succeed) reported by the AOP module. Finally, in the case of the remainder 2 questions regarding the use of the course's CMS or GitHub, around 90% prefer GitHub to the CMS to distribute the template assignments (the remainder 10% keep neutral), while 77% prefer GitHub to the CMS (8%) to hand in the assignments (being 15% neutral).

D. Teachers' perceptions

While setting up the course in *GitHub Classroom* and its connection to *Travis* constituted an easy task thanks to the available documentation [20], [21], most of the effort spent by the instructor in the beginning was devoted to configuring each assignment's repository. More specifically, we need to configure each assignment's repository to setup the corresponding database in *Travis*; while *Travis* gives support for a wide number of databases (e.g., MySQL), for other databases (e.g., Oracle), we need to explicitly install and configure them. Additionally, we devote extra time to design quality DbUnit tests [16] for each assignment, task which requires, for example, to setup and teardown databases, hand special datatype values, or apply performance tricks to speed up tests. We note that, once we have configured repositories in both MySQL and Oracle, and designed DbUnit tests, we can reuse some of these details to get a head start for next assignments (for example, assignments that share a database will use the same database configuration in *Travis* and the same setup and teardown aspects in DbUnit tests).

As for time spent on feedback and grading, we do not have registered data from previous years, so we can not provide an objective comparison with previous years, but give our perceptions. We have experienced a significant reduction in working times, specially in grading. As such, per each assignment, the teacher can check both if integration tests are passed and if best practices are adopted, just by accessing *Travis* and taking a quick glance at its feedback. It usually takes around 1 minute. Depending on the previous results, the teacher can decide to manually inspect the code on GitHub, which usually takes another 2-3 minutes of additional time. With the previous system, this process used to take around 20 minutes (we have to download the assignment from the CMS, import it into Eclipse, configure the local database, run tests, manually inspect the code to check BPs, etc.). Thus we have observed a significant reduction of time.

V. Related Work

Our proposal for enhancing JDBC learning mainly relies on adopting CI techniques, and providing automatic formative assessment based on integration tests and AOP.

On the one hand, the implementation of modern industry practices, such as CI, in SE courses is not new. Focusing on programming courses, several works have used CI tools to teach (mainly) basic programming courses, with motivating outcomes, some of them common to ours, such as improving the management process of students' assignments and efficiency of deliveries [3], [4], [29]–[33], providing real-time formative [3], [4], [29], [32]–[34] or summative [3], [4], [29], [33] feedback, streamline grading of students' tasks [3]–[5], [29], [33], [35], improving students' grades [4], [31] or improving collaborative learning and project management [34], [36], [37], while exposing students to industry tools.

On the other hand, a large number of approaches to automated program assessment can be found in the literature (e.g., see surveys [15], [38], [39]), which mainly differ on the kind of feedback they provide. Based on the types of feedback components identified in [38], our proposal mainly focuses on providing *knowledge* about both *mistakes* and *how to proceed*. In the former case, we inform about *solution errors*, such as runtime errors that are reported by the AOP module, and *test failures*, shown by the integration tests. In the latter case, our AOP module mainly provides hints and solutions for "error" correction (that is, what is needed to be done to adopt a not implemented BP). To the best of our knowledge, our approach is the first one that provides automatic formative feedback to help students adopt programming best practices, and uses AOP (commonly used for system monitoring purposes) to give such an assessment, encouraging and motivating students to implement best practices. As for the *technique* to generate feedback [38], our proposal, as well as the previous cited CI-based solutions that provide formative assessment [3], [4], [29], [32]–[34], bases on *dynamic code analysis*, using existing test frameworks for *automated testing*. However, while those works use JUnit, for unit tests, we use DBUnit, for integration tests, which becomes a bit of challenge as shown previously.

VI. Discussion

In this section, we discuss fine-grained aspects of the proposal, highlighting findings and lessons learned (in bold text), and limitations (in italic text). We have organised this discussion around four different aspects.

A. Assignments' management and grading, and industrial tools' familiarization

Preliminary results show that the intervention benefited all parties involved. From the students' feedback, we can conclude that **the students value positively the use of *GitHub* not only for storing and handing in their assignments (Q1/Q2), but also for their personal projects (Q4)**. Even, they prefer this tool to the course's CMS to distribute and hand in assignments; while delivering assignments just requires accepting the corresponding teacher's GitHub invitation, handing in an assignment is reduced to commit the work from the Eclipse's Git perspective. In contrast, *results shown that most of students pay little attention to Travis feedback* (Q5). Our personal perception is that, having been developing and testing their projects using Eclipse, students probably are used to check their feedback in the Eclipse console view. Thus, they

would not have any need for checking it again through *Travis*. Regardless, **students state that the proposal helps them get familiar with both industrial tools**.

From teachers' point of view, although *some effort was spent in the beginning to configure assignments*, **the benefits in terms of efficiency are significant**, both when students ask for help and when teachers have to grade students' assignments. In both cases, *Travis* feedback and *GitHub* online access to students' code have proven to be very useful. In this regard, so that teachers can check last version of students' work in both situations, *students must get used to have previously committed their code* (which sometimes is a real challenge, thus teachers have to be continuously reminding students of this fact). We note that, *with the proposed system, student work is not on University owned servers (in contrast to commercial CMSs), which might represent a limitation.*

B. Grading formative activities

There has been quite a bit of controversy about grading formative assessment activities or not. While some works suggest that the evaluative aspect of grading may distract students from a focus on learning [40], other authors argue that low stakes assessments encourage students to pay more attention [22]–[24]. As mentioned above, the grade assigned to the course's laboratory assignments is of minor importance, and is clearly designed to motivate students and acknowledge their effort. However, *when tasks are graded, some students may decide to copy other students' work* [22], [41]. If undetected, plagiarism can facilitate students' progress through courses without achieving the desired learning goals [41]. It makes of detecting plagiarism an ongoing challenge each academic year. In particular, we use the *MOSS* tool to detect cheating on students' assignments [41]. We have to recognise that, **so far, no incidence has been observed in the students, who have shown responsibility in their assignments**.

C. Usefulness and quality of the feedback

Focusing on the feedback itself provided by our test suite, we find it useful that **students can benefit from rapid feedback loop, instead of waiting for work to be graded**. Additionally, the feedback report helps them gain an understanding of their work: (i) failing integration tests provide a hint or guidance on what might be wrong, so students can iteratively improve on their work, and (ii) AOP feedback provides rewarding messages highlighting adopted BPs (green), hints regarding not adopted BPs (yellow) and execution errors (red). In fact, students reported that **AOP feedback has motivated them to give more attention to the adoption of JDBC BPs**.

As for the quality of automatic feedback, teachers often agree that it is not possible to assess automatically all the issues related to good programming [15]. Although not many, *there are some aspects that are not checked by our AOP strategy, such as the use of optimized SQL instructions or the creation of resources outside loops*. We consider these aspects not explicitly related to JDBC knowledge, but to prior knowledge students are expected to gain from previous courses. In any case, *to provide a higher quality feedback, the teacher can combine the automatic assessment provided by our proposal with a manual inspection of the code*, looking for such few, not checked, aspects. Overall, given these results and considering students' feedback, our vision is that **our AOP-based approach could also be applied to other courses** which teach, for example, programming BPs or finite resources' management (e.g., connections, threads, or files). As for technological aspects, **the unique requirement for using our approach is to have an AOP implementation compatible with the programming language taught in the course**, which we do not consider a hindrance given the large number of languages that implement AOP (e.g., PHP, Python or C#) [42].

D. Students' scores

Although *the experiment has been applied only during an academic year*, which could be considered a limitation, the results obtained suggest positive effects; **not only have we detected a significant improvement regarding the adoption of JDBC best practices** by the students attending the course in this academic year 2019/20, **but we have also experienced statistically significant differences between the exam scores of the analyzed academic courses**. Thus, we can conclude that, far from leading to a detrimental effect on students' JDBC marks, the experience has contributed to an improvement in the academic performance.

Based on the highlighted results, we can say that the main intentions that led to the introduction of the proposed framework are achieved, which motivates us to continue with this proposal.

VII. Conclusions

We have presented our experiences in developing and running a framework to enhance the learning of JDBC, combining CI, mainly to manage and grade students' assignments and support the execution of integration tests, and a novel application of AOP, mainly to encourage students to adopt JDBC best practices. In the particular case of CI, although it has already been used in teaching (mainly) basic programming courses which use unit test, we have to face with integration tests, which brings more complexity to testing design and management. The proposal is underpinned by an academic year experience, and by an evaluation using objective (students' marks and tests' logs), and subjective data (students' feedback and instructor's perceptions). The results drawn from this experience show that the six main motivating intentions that led to the introduction of this intervention, are achieved.

As future work, we plan to extend our framework to cover ORM programs, as the other technique taught in the course to implement the *persistence* layer. As advanced previously, we also intend to examine the possibility of applying our proposal to other courses.

VIII. Data Availability

The data that support the findings of this study are openly available in its supplementary material [19].

REFERENCES

[1] D. Oguz and K. Oguz, "Perspectives on the gap between the software industry and the software engineering education," *IEEE Access*, vol. PP, pp. 1–1, 08 2019.

[2] G. B. Ghantous and A. Gill, "Devops: Concepts, practices, tools, benefits and challenges," in *Proceedings of the 21st Pacific Asia Conference on Information Systems (PACIS'17)*, 2017, p. 96.

[3] V. Gennarelli. (2017) Real-time feedback for students using continuous integration tools. Available at https://github.blog/2017-03-01-real-time-feedback-for-students-using-continuous-integration-tools/. Accessed January 2021.

[4] ——. (2019) How GitHub Classroom and Travis CI improved students' grades - The GitHub Blog. At https://github.blog/2019-02-12-how-github-classroom-and-travis-ci-improved-students-grades/. Accessed January 2021.

[5] E. Kral and P. Capek, "Towards using continuous integration tools to teach programming courses," in *Proceedings of International Conference on Computational Science and Computational Intelligence (CSCI'15)*, 2015, pp. 871–872.

[6] F. Maydene, J. Ellis, and J. Bruce, *JDBC API Tutorial and Reference (Third Edition)*. Addison-Wesley Pub Co, 2003.

[7] EDUCBA. (2021) Difference between Unit Test vs Integration Test. Available at https://www.educba.com/unit-test-vs-integration-test/. Accessed January 2021.

[8] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, and J. Irwi, "Aspect-oriented programming," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97)*, Berlin, Heidelberg, 1997, pp. 220–242.

[9] The O'Reilly Authors, *Java Enterprise Best Practices. Expert Tips & Tricks for Java Enterprise Programmers*. O'Reilly, 2010.

[10] J. C. Gutjahr and A. Loew, "Scalability and Performance: JDBC Best Practices and Pitfalls," in *Proceedings of the Net.ObjectDays Workshops*, 2002, pp. 449–463.

[11] J. Biggs, "Enhancing teaching through constructive alignment," *Higher Education*, vol. 32, pp. 347–364, 10 1996.

[12] B. S. Bloom, "Some theoretical issues relating to educational evaluation. I RW Tyler (Red.), Education evaluation: New roles, new means." 1969.

[13] B. S. Bloom, J. T. Hastings, and G. F. Madaus, *Handbook on Formative and Summative Evaluation of Student Learning*. McGraw-Hill Book Co, New York, 1971.

[14] M. Perie, S. Marion, B. Gong, and J. Wurtzel, "The role of interim assessments in a comprehensive assessment system: A policy brief," *Dover, NH: National Center for the Improvement of Educational Assessment.*, 2007.

[15] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83–102, 2005.

[16] DbUnit. (2021) Version: 2.7.1-SNAPSHOT. Available at http://DbUnit.sourceforge.net/. Accessed January 2021.

[17] G. Kiczales and M. Mezini, "Aspect-oriented programming and modular reasoning," in *Proceedings of the 27th International Conference on Software Engineering (ICSE'05)*. New York, NY, USA: ACM, 2005, pp. 49–58.

[18] R. Laddad, *Aspectj in action: enterprise AOP with Spring applications*. Manning Publications Co., 2009.

[19] Supplementary Material. (2021) Enhancing the Learning of Database Access Programming using CI and AOP. Available at https://zenodo.org/record/4434507.

[20] A. Jones. (2021) Set up your digital classroom with github classroom. Available at https://github.blog/2020-03-18-set-up-your-digital-classroom-with-github-classroom/. Accessed January 2021.

[21] Education Travis CI. (2021) How can Travis CI help with your studies? Available at https://education.travis-ci.com/. Accessed January 2021.

[22] I. J. M. Arnold, "Cheating at online formative tests: Does it pay off?" *Internet Higher Education*, vol. 29, pp. 98–106, 2016.

[23] A. B. G. Alexandron, M.E. Wiltrout and J. Ruipérez-Valiente, "Assessment that matters: Balancing reliability and learner-centered pedagogy in mooc assessment," in *Proceedings of the Tenth International Conference on Learning Analytics Knowledge (LAK'20)*, 2020, pp. 512–517.

[24] S. Schut, J. van Tartwijk, E. Driessen, C. van der Vleuten, and S. Heeneman, "Understanding the influence of teacher–learner relationships on learners' assessment perception," *Internet Higher Education*, vol. 25, p. 441–456, 2019.

[25] J. Cohen, *Statistical power analysis for the behavioral sciences*. Routledge, 1988.

[26] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes (1st edition)*. Springer-Verlag Berlin Heidelberg, 2011.

[27] P. Reimann and K. Yacef, *Using process mining for understanding learning*. Handbook of design in educational technology. New York: Routledge, 2013.

[28] M. Song and W. M. P. V. der Aalst, "Supporting process mining by showing events at a glance," in *Proceedings of 17th Annual Workshop on Information Technologies and Systems (WITS'07)*, 2007, pp. 139–145.

[29] S. Heckman and J. King, "Developing software engineering skills using real tools for automated grading," 2018, p. 794–799.

[30] K. L. Reid and G. V. Wilson, "Learning by doing: introducing version control as a way to manage student assignments," in *Proceedings of the 36th ACM Technical Symposium on Computer Science Education (SIGCSE'05)*, 2005, p. 272–276.

[31] M. L. Pilla, "Teaching computer architectures through automatically corrected projects: Preliminary results," *International Journal of Computer Architecture Education*, vol. 6, pp. 62–67, 2017.

[32] C. Matthies, A. Treffer, and M. Uflacker, "Prof. CI: Employing Continuous Integration Services and GitHub Workflows to Teach Test-Driven Development," in *Proceeding of IEEE Frontiers in Education Conference (FIE)*, 2018, pp. 1–8.

[33] C. H. Tran, "Applying test-driven development in evaluating student projects," 2 2020, faculty of Science and Engineering, Department of Information Technologies.

[34] C. Hsing and V. Gennarelli, "Using github in the classroom predicts student learning outcomes and classroom experiences: Findings from a survey of students and teachers," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education, (SIGCSE'19)*, 2019, pp. 672–678.

[35] Y. Cai and M. Tsai, "Improving programming education quality with automatic grading system," in *Proceedings of the Second International Conference of Innovative Technologies and Learning (ICITL'19)*, 2019, pp. 207–215.

[36] C. Z. Kertész, "Using github in the classroom – a collaborative learning experience," in *Proceedings of the 21st International Symposium for Design and Technology in Electronic Packaging (SIITME'15)*, 2015, pp. 381–386.

[37] J. Sus and W. Billingsley, "Using continuous integration of code and content to teach software engineering with limited resources," in *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, 06 2012, pp. 1175–1184.

[38] H. Keuning, J. Jeuring, and B. Heeren, "Towards a systematic review of automated feedback generation for programming exercises," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'16)*. ACM, 2016, pp. 41–46.

[39] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming: A review." *ACM J. Educ. Resour. Comput.*, vol. 5, no. 3, p. 4–es, 2005.

[40] T. Wolsey, "Efficacy of instructor feedback on written work in an online program," *International Journal on E-Learning*, vol. 7, no. 2, pp. 311—-329, 2008.

[41] J. Pierce and C. Zilles, "Investigating student plagiarism patterns and correlations to grades," in *Proceedings of the 48th ACM Technical Symposium on Computer Science Education, (SIGCSE'17)*, 03 2017, pp. 471–476.

[42] Y. Lilis and A. Savidis, "A survey of metaprogramming languages," *ACM Comput. Surv.*, vol. 52, no. 6, Oct. 2019.