Integration Model for Learning Management Systems, Source Control Management, and Autograders using Service-Oriented Architecture Principle

Bram Musuko Panjaitan
School of Electrical Engineering and Informatics, Institut Teknologi Bandung
Bandung, Indonesia
Email: 13517089@std.stei.itb.ac.id

Satrio Adi Rukmono
School of Electrical Engineering and Informatics, Institut Teknologi Bandung
Bandung, Indonesia
Email: sar@itb.ac.id

Riza Satria Perdana
School of Electrical Engineering and Informatics, Institut Teknologi Bandung
Bandung, Indonesia
Email: riza@informatika.org

***Abstract*—Most learning management systems (LMS) use a file uploader that receives archived source code from a student for programming exercises and requires the teacher to grade it manually. In this approach, students do not learn to use standard professional tools to work on a source code, and the teachers also spend much time grading. There is an opportunity to use source control management (SCM), such as Git via GitHub or GitLab, as a submission method for students. This mechanism helps programming students practice a common process used in the professional world as early as possible. Rather than manual grading, autograders are widely used in Learning Management Systems to help instructors grade student works. Autograders work faster than humans and provide objective grading. This paper discusses an integration model for learning management systems, source control management, and autograders, each of these components is usually used separately. We write a reference implementation that uses Moodle as the LMS and GitLab as the SCM. We also build a minimally functional autograder in place for proof-of-concept in this implementation. Students can submit their work using the merge request feature provided by GitLab from the repository that they fork from the instructor's original repository. The system captures the merge request event, and the autograder starts grading student works and updates student scores in the LMS. We also discuss how the system performs when dealing with many requests semi-simultaneously to simulate an exam situation. The system follows the Service-Oriented Architecture (SOA) principle to keep each component agnostic, and developers can use any LMS, SCM, and autograder they find suitable. In our experiment, the system can handle 200 submissions in a short period amount of time. The results are that the student learns SCM basic workflow using the system, and the teachers are helped by automated grading.**

***Index Terms*—integration, learning management system, autograder, webhook, source control management**

I. Introduction

Teaching programming students to use best-practice technology helps them prepare for the professional world. There are many opportunities to put a set of best-practice methods into their learning process. For example, one best practice that can be taught is to use source control management when sharing code.

A lot of educational organizations use a Learning Management System as a platform to carry out learning activities. In operating programming learning activities, many Learning Management Systems use an old-fashioned way rarely used in a professional setting to submit student source code work. Some Learning Management System still requires students to archive their code into a zip or rar format and upload their code using the LMS' file uploader. There is room for improvement in this process. The uploader process can be replaced with source control management so that students directly practise a common professional task of "publishing" source code while learning to program. This new experience will pretty much be the same as when a developer pushes their code into a source control management and trigger a CI/CD pipeline to test the code, build, and deploy the resulting artefact. However, instead of CI/CD, this push-action triggers the grading process and update the student's score in the learning management system. This paper discusses how to integrate a learning management system with source control management and autograders.

II. Foundations

A. Learning Management System

Learning Management System (LMS) is an information system used to support operating digital learning. Learning Management System will help education organizations to share and distribute learning content, provide an administration system, provide a communication channel for instructors and students. A good Learning Management System follows a few criteria, such as content can be changed easily to adjust the requirement, have an automatic and centralized administration system, and show lesson material quickly [1].

B. Source Control Management

Source Control Management (SCM) is a system that help developer to track and manage every change on their codebase. One of the most used SCM is Git, an open-source distributed version control system [2]. One of Git's features is branching, duplicating a version of code into an independent branch. GitLab is a web-based DevOps lifecycle tool that provides a Git repository manager [3]. GitLab adds features on top of Git, such as GitLab Webhook and GitLab Forking. GitLab Webhook is a developer-defined HTTP Callback in the GitLab repository. This callback is triggered when some events occur, including merge requests, push, and wiki events. It then sends a POST request to the destination URL with information related to the event attached to the body. Developer can define their destination URL for GitLab Webhook. GitLab Forking is a mechanism to duplicate a repository from any repository to a new personal repository. This new repository will bring all of the commit histories from the original repository. Developers can use the forking mechanism to create their custom version or develop a new feature. When developers finish a new feature, they can add it to the original repository using a merge request from their repository to the original repository.

C. Autograder

Autograder is a tool to assess source code automatically with a predefined method. Thus, autograders reduce the amount of time required to assess students' programs and provide objective results. An autograder can be implemented using a black-box or white-box approach. A sandbox environment, separate from the host machine and execution environment, is needed for executing a source code in an autograder. A sandbox environment limits CPU, memory, network, and other resources to prevent abusive programs from impacting the operating system.

In general, there are two ways to achieve a sandbox environment. Those methods are containerization and jailed sandbox. In containerization, the server will create a container on top of the host OS. This container has a dependency that might differ from the host machine and has limited access, such as read-only on a particular folder. The server will execute the source code inside the container, receive the execution's output, then destroy the container. One of the popular tools that can be used to manage container lifecycle is Docker. The second method is jailed sandbox. This method is done by creating an untrusted user with limited authority, and then this user executes the source code directly in the server. Peveler et al. compare jailed sandbox and container performance when creating a sandbox environment [4]. They point out that jailed sandbox has a slightly faster execution time (∼0.6 seconds) than a container. This difference happened because Docker needs to create and destroy the container, which takes about ∼2.4 seconds and can be improved by using a Docker pool and destroying the container in the background. On the other hand, it is harder to achieve a wholly separated dependency between the host machine and the execution environment with a jailed sandbox than a container.

III. Integration Model with Bridge Service

Using best-practice technology in the student learning process is still uncommon. This will impact students, as they may not have enough experience to operate the technology when entering the professional world. Meanwhile, if the best practice technology is used while learning, not only do students learn to use the technology, but instructors can also keep adapting to the best practice technology. For example, in learning to program using the LMS context, source control management can replace the mechanism of doing submission using an uploader. Also, a manual grading that instructors do to assess student source code works is repetitive and exhaustive work that an auto-grading tool can help. Based on that problem, we will build a learning management system integrated with source control management and an autograder. Integrating a learning management system with source control management means that an assignment in LMS can be solved by submitting work in SCM. This approach can be realized by utilizing Webhook in SCM. The system will pick student submission and process it to get the score and update the data in the LMS. An integrated system with an autograder means that the system can use the autograder to produce a score from student submission. After the system gets the student submission, an autograder will be executed to assess student work.

The solution applies Service Oriented Architecture as a design guideline to break down the service component. Service-Oriented Architecture is a paradigm to decompose a big problem into several small units of encapsulated, integrated services [5]. Some of SOA characteristics are open standard, loosely coupled, support different vendors of technology. If organizations design a good SOA, they get to benefit from the scalability aspect. Scalability is a system's capability to handle different loads [6]. One SOA design pattern is queue [7]. By using a queue, servers can communicate asynchronously. The server will put the message inside the queue, and the target service will try to pull the message from the queue. When replying to the message, the target service can use one of these two options. The first option is to also reply to the message using a queue, so the communication is still asynchronous. The second option is to use a callback, and the communication is synchronous.

Bridge service will be used to integrate LMS with SCM and Autograder. The bridge service's goal is to have easier integration across services. A system administrator can quickly change other components using a bridge service, such as a different LMS or any SCM or grader. Bridge service will adjust the API contract to match the contract between LMS, Grader, and SCM and the adjustment done in the LMS, grader, and SCM is minimal. Therefore each component in this system will become agnostic. Bridge service will have asynchronous communication with graders because graders need some time to assess source code. The communication uses a queue from the bridge service to the grader. The grader can reply synchronously using a callback. Grader workers need to execute student source code in a sandbox environment to limit resource availability and secure the host machine from malicious code. This environment can be done with the help of a Docker container.

Based on the above discussion, we devise requirements as follows.

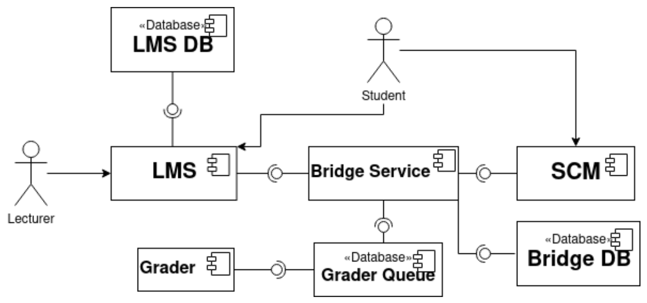

Fig. 1: Interaction Between Components

- The instructor can create a programming assignment, and the system provides an SCM repository link for student to submit their work.
- The instructor can upload a metric file when creating a programming assignment. The autograder uses these metric files to assess students' source code.
- Student can link their SCM account into LMS. The system can recognize student submission from SCM to update student data in LMS using this data.
- Student can submit their work using merge requests in SCM. This event must be captured by the bridge service and passed to the autograder to perform the grading.

The communication between each component needs to apply the open standard principle from SOA. For example, when bridge service helps LMS generate the SCM repository link, each component's request and response can use an open standard format, such as JSON format. The system also needs to support different vendors of technology principles in SOA. Each component can have a different technology stack. Each component also needs to be loosely coupled and independent from the other. If there is any improvement that needs to be done for teachers in LMS, other components will have minimal or no changes.

Aside from the functional requirements, the system also needs to consider some non-functional aspects. For example, the system needs to keep available when many students submit their work in a short amount of time. This situation frequently happens when there is a deadline for an assignment or an exam. Most of the students will submit their work close to the assignment's deadline or the end time of the exam. The system must be able to receive all those requests and to the grading process. The system has multiple grader workers that can be added/removed accordingly to adjust the system's load to handle this scenario. Based on that aspect, the non-functional requirements are as follows.

- Scalability—the system can be scaled to face a different number of requests.
- Throughput—the system can work with a certain large number of users.

There are four main components shown in Fig. 1, LMS, SCM, Bridge Service, and Grader Worker. Bridge service and LMS has their database. A queue will be used between bridge service and Grader workers. Next, we discuss each process.

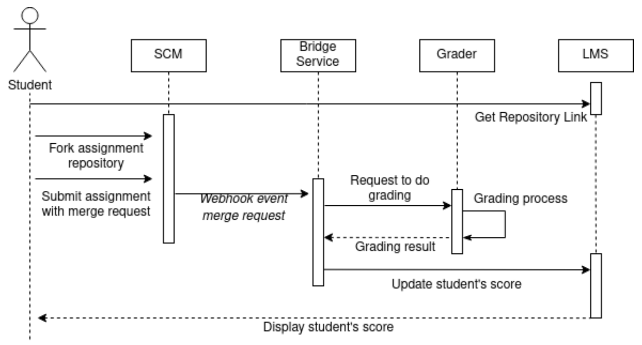

Fig. 2: Sequence Diagram - Student assignment workflow

a) Creating a programming task: The process starts when the instructor opens the task creation form on the LMS page. There, the instructor fills in assignment data and upload a metric file. After LMS creates the assignment, LMS will create a request to bridge service to create a repository in SCM using SCM API. After that bridge service will save the repository link and give LMS the link so that the LMS can show the link for students and instructors.

b) Connecting an SCM account with the LMS: This process is achieved with the help of the bridge service. A student needs to press the verification button inside the LMS profile page and be redirected to the bridge service page. Bridge service will also redirect the student to the SCM login page. After students log in successfully, the bridge service will get user data from the SCM, such as ID and username. The data will be saved in the bridge service database and LMS database.

The sequence diagram is shown in 2. Students can go to the LMS and look at their SCM repository assignment link, and then they can fork the central repository and then commit and push the changes into SCM when they finish the task. Then, the student can create a merge request from their repository to the central repository for that assignment to submit their work. When a merge request is issued, SCM will send a webhook to the bridge service. Bridge service will validate the webhook, get student source code, student data, and metric file, and send it to an autograding queue. After the autograder finishes the grading process, the bridge service receives a callback with the student's score and forwards the score to the LMS using the API provided by the LMS.

An autograding task starts when the bridge service sends a grading request through the queue. Autograder will start the work by extracting the student's source code to a temporary location. After that, a container will be spawned to execute student source code. This execution is run for all the test cases. After all the test cases have been evaluated, the autograder summarises the student store and send it to the bridge service.

IV. Reference Implementation

A. Implementation

In this implementation, LMS, SCM, and an autograder are integrated with a bridge service. Moodle is used in this implementation because of its excellent flexibility for customizability. Also, Moodle has an open-source license and a vast developer community. GitLab is used as the SCM because of its open-source nature, while the bridge service is built using NodeJS and express web framework and PostgreSQL database management system. The bridge service database includes four entities: student, assignment, metric file, and submission history. The bridge service has three integration points: with Moodle, GitLab, and an autograder. The bridge service helps Moodle in GitLab account verification, creating a new assignment, and updating user scores. When verifying the account, the bridge service will provide GitLab OAuth login, get GitLab username and GitLab id, save it to the bridge service database, and send it to Moodle using core_user_update_users function in Moodle external API. When creating a programming assignment, the bridge service will save assignment detail and store file metric inside a file server in the bridge service and request GitLab using GitLab API to create a repository and pass it to Moodle. After the bridge service gets the score from the grader, it updates the score in Moodle using core_grades_update_grades function, mod_assign component in Moodle external API.

Bridge service integrated with GitLab when providing GitLab OAuth, creating a repository, and receiving webhook. GitLab login OAuth is achieved by using the passport and passport-gitlab-2 library. Bridge service needs application id and application secret to use this library and provide GitLab OAuth. This login is only done once the user verifies for the first time to get the username and GitLab id. Other token-related data will not be stored or taken by bridge service. When creating a programming assignment bridge service, use GitLab API with @gibreaker/node library as GitLab API wrapper to keep repository name unique, assignment id included in repository name, bridge service also creates GitLab webhook that connect that repository to bridge service with merge request event only. A private token instructor is hardcoded inside the bridge service to use this API. When GitLab notifies that there is a merge request event, the bridge service will take student source code using GitLab API and create an archived file from a list of files. For integration with the grader, the bridge service needs to request grading. This process can be achieved by sending the message via RabbitMQ as a queue. This message contains student source code, metric file, and student data. The grader will reply via callback, and the bridge service will provide that endpoint. When this happens, the bridge service will save the submission history and send the result to Moodle.

Moodle will have a slight modification to support a new programming assignment and GitLab verification. When an instructor creates a new assignment, two new fields need to be filled, the first is the metric file field, and the second is the grading method. The Metric file field is used to get the metric file for programming assignments and the grading method to choose whether the grading will do once for the first submission only or every submission as long as the due date has not crossed yet. These added fields can be achieved by modifying Moodle assignment creation form module. To support GitLab verification, the user profile field in Moodle added an extra field: GitLab. GitLab field in user profile has two attribute username and isGitlabVerified. A student cannot fill this field manually. Instead, they need to press a button created inside the edit profile page that redirects the user to the bridge service page. We can add this button by modifying the *edit profile form* module in Moodle.

A simple black-box autograder is built for this implementation using the Python programming language for grading Python programs. The grader connects with RabbitMQ using the pika library. The grader consumes the message from the queue and only acknowledge the message when the grading process is finished to prevent any data loss if there is an error while doing the grading process. When it receives a grading request from the queue, it parses the message and retrieves the student's source code encoded in base64. The code is then extracted into a temporary folder using the shutil and tarfile libraries in the host machine. Next, the grader will spawn a docker container using the docker-py library and execute student source code. Finally, the grader will execute a docker container for each test case in an array of test cases.

```
self.container = client.containers.run(
    image="python:3.9-alpine",
    command=f"sh␣-c␣'{command}'",
    read_only=True,
    network_mode="none",
    volumes={self.tmpPath: {"bind": "/workspace",
            "mode": "ro"}},
    working_dir=os.path.join("/workspace", "src"),
    nano_cpus=1 * 1000000000,
    mem_limit="128m",
    memswap_limit="256m",
    pids_limit=64,
    detach=True,
    log_config={
    "config": {
        "mode": "non-blocking",
        "max-size": "1m",
        "max-file": "2"
    }}
)
```

Fig. 3: Docker container configuration

Docker container configuration is shown in Fig 3. Docker will use python docker image because, in the testing, the student will use python language. Network, CPU, memory, memory swap is limited in the docker container. Volume in the docker container binds with the host machine so that that docker container can access student source code. Each test case will be piped into the python3 executable. After finishing the grading, the grader will send the total score to the bridge service with a callback URL, send an acknowledgement to RabbitMQ and be ready for the following message.

No modifications are required in GitLab for the integration. Therefore, we use the main hosted https://gitlab.com in our reference implementation. In GitLab, there are some tokens needed. The first one is a private token. This token is used for creating a repository and fetching student source code. The

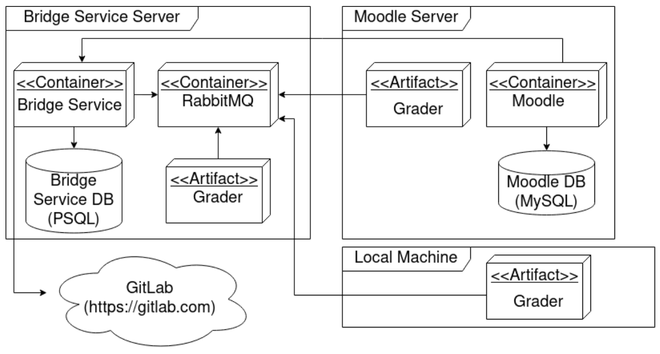

Fig. 4: Deployment Diagram

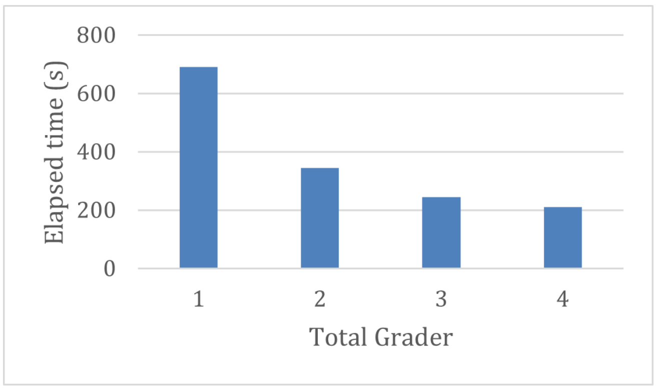

Fig. 5: Simulation time comparison

second token is application secret and application id. These two tokens are required to create GitLab login OAuth.

The deployment diagram is shown in Fig. 4. Three environments were used, those are bridge service server, Moodle server, and local machine. Bridge service server and Moodle server use Digital Ocean cloud VPS service. The bridge service server uses 1 GB RAM and 1 CPU, while the Moodle Server uses 2 GB RAM and 1 CPU. There are some components in the bridge service server: bridge service container, RabbitMQ container, bridge service database using PostgreSQL with a container, and grader worker. Moodle server consists of Moodle container using moodlehq/moodle-docker container with MySQL container as Moodle database and grader worker. Grader workers are also deployed on a local machine. Grader worker is deployed in many places to speed up the whole process required to do the grading. The grader worker is not in a container because the grader worker will spawn a docker container when doing the grading process. If the grader spawns a docker container inside a docker container, extra complexity can be avoided by just running the grader worker outside the docker container.

B. *Testing*

Testing is carried to check whether the implementation is already aligned with the requirement mentioned previously. There are three phases of testing executed: functional testing, non-functional testing, and student-facing testing.

Functional testing held to check the implementation already support all functional requirement. There are three scenarios for this testing. The first scenario is a teacher creating a programming task and uploading a metric file, checking if Moodle and bridge service saves the metric file and creating a GitLab link. The second scenario is a student doing a GitLab account verification from Moodle page and checking if the data is saved in the database. The student can also see their GitLab username from Moodle. The final scenario is a student submitting source code using a merge request and checking if their score is posted on Moodle assignment page.

For the first scenario, the system successfully creates and stores a programming task with a metric file and provides a GitLab repository as the central repository for that assignment. The system successfully saves the GitLab and Moodle student association data in the database for the second scenario. Moodle also show GitLab username in Moodle profile. For the last scenario, the system successfully captured the merge request event and did grading using autograder. Bridge service also updates student scores in Moodle. Thus, all the functional requirement has been achieved.

Non-functional testing is held to check the implementation is supporting non-functional requirements, those are availability and throughput. The author simulates an exam situation where there are many submissions in a short time. This execution will be run four times with the different grader workers, so there is a condition where the system has one grader, two graders, three graders, and four graders. There is 200 submission request that sent in 3 minutes. Those requests will test the throughput of whether the system can handle all those requests and test the scalability whether the system can handle all requests faster when there is an extra grader added. The first two autograders are deployed in the cloud, with 1GB RAM and 1 CPU for the first grader and 2GB RAM and 1 CPU for the second. The third autograder is deployed on the local machine with 16GB RAM and 4 CPUs. The fourth autograder is also deployed on the local machine with 8GB RAM and 8 CPU. To simulate 200 submission requests, the author uses 4 GitLab account that submits their work 50 times for each account and using a script to submit each submission subsequently with a total time of 3 minutes to send all those 200 requests with average speed 200/180 second (1.11 second).

Based on Fig 5, there is an improvement in the total time needed for the system to finish all the requests with an extra grader. Therefore, more graders result in faster elapsed time, but there will be a plateau if there are too many graders. Aside from the total grader number, machine specifications also determine the speed of execution time. Based on Table I, the local environment has a faster average execution time due to superior resources. The local environment uses a four-core CPU and 16GB RAM. The Local Environment (Laptop) uses an eight-core CPU and 8GB RAM.

When the system uses only one grader, the elapsed time is so high. This longer time happened because the message comes with a speed rate of 1.11 seconds is faster than the average execution time that the bridge service server has, which is around 3.62 seconds. Therefore, there will be many messages in queue piling up and waiting for the grader to finish the grading process. Fewer messages in the queue pile up when extra graders are added, so the total elapsed time will be faster. However, when there is too much grader, one of the graders will not get the grading request because other graders will consume it first, so the elapsed time will become a plateau.

TABLE I. Average and standard deviation grader execution time

Name	**Avg execution time (s)**	**Std execution time (s)**
Moodle server	4.0	0.18
Bridge Service server	3.62	0.24
Local Environment	1.51	0.24
Local Environment (Laptop)	2.03	0.21

Student-facing testing was conducted to ensure that the student could use the system smoothly and acquire potential feedback. Students are asked to verify their GitLab accounts on the Moodle page and submit their work using merge requests. The account verification test runs smoothly without any hurdle. However, students need to learn to use Git features such as git commit, push, add, and clone for submission using merge request. They also need to familiarize themselves with GitLab's fork repository and merge request features. In this testing process, we received feedback that there needs to be instruction on submitting source code using this new method in the LMS.

V. Discussion

We can see that the design's implementation is working according to the requirement. Using SOA also helps speed up the system's implementation because each component will be independent. In addition, open-source applications such as Moodle and GitLab can be used for our model.

One of the flaws of using merge requests in SCM to submit students' works is that other students can see the merge requests. Thus, this merge request method can lead another student to cheat by looking at other students' work and copying the code. The easy solution to tackle this issue from the author is to add one extra step in the grading process. When there is a merge request, the bridge service will receive the event and directly delete the merge request. Then the bridge service will fetch the code from the student's repository directly.

Nevertheless, this approach also has some disadvantages. The first one is that it will be hard for a teacher to do manual checking since the merge request is deleted. The second one is that there will be a fraction of time from a student submitting work to bridge service deleting the merge request. There is no guarantee that other students do not access other student merge requests in this tiny time gap.

Related work has been done by Inggriani Liem and Karol Danutama in 2013. The similarity is integrating Moodle with an autograder. For the architecture, they use a service dispatcher to control the queue load that graders will consume. The big difference between our research and the previous work is the submission method. The previous research creates a new question type in Moodle called "source code" for students to upload their archived source code in the uploader. Meanwhile, in this research, source control management is used for students to submit their work.

VI. Conclusion

Based on the implementation and testing, we can conclude that the Learning Management System is integrated with Autograder and SCM with the help of the bridge service as an integrator service. Using SCM in the learning process can teach a student to use the Git command and SCM feature. Automated grading also helps the teachers. If we look at the testing performance, the time system needs to handle 200 requests in 3 minutes is 3 minutes and 30 seconds, which is relatively fast. The functional and non-functional requirements that define the design process is all implemented in the reference implementation. The system also decomposed to several small services, so it is easier to change components in the future if needed.

The author has some suggestions that can be considered for future works. First, a configuration in Moodle page when teachers create a programming assignment can be more detailed so that each assignment can be more customizable. Second, a worker or a queue can be added inside bridge service to handle some bridge service heavy tasks and add more throughput in bridge service. Third, the bridge service can be split into several services to better follow SOA principles. Lastly, the cheating aspect needs to become a consideration. As we mentioned, in our current setup, students can see another merge request. For the system to be used in an actual education setting, this will need to be addressed.

References

[1] Ellis, R. K. (2009). Field Guide to Learning Management Systems. ASTD Learning Circuits.Git

[2] GitLab. 2020. The First Single Application For The Entire Devops Lifecycle - Gitlab. [online] Available at: ¡https://gitlab.com/¿ [Accessed 17 June 2021].

[3] Git-scm.com. 2020. Git. [online] Available at: ¡https://git-scm.com¿ [Accessed 22 August 2021].

[4] Danutama, K., & Liem, I. (2013). Scalable Autograder and LMS Integration. Procedia Technology

[5] Erl, T., (2005). Service-Oriented Architecture: Concepts, Technology, and Design. 1st ed. Prentice Hall.

[6] Abbott, M. L. & Fisher, M. T,, (2015). The Art Of Scalability. 2nd ed. Addison-Wesley.

[7] Erl, T., (2009). SOA Design Patterns. 1st ed. Prentice Hall PTR.