DevOpsEnvy: An Education Support System for DevOps

Guoping Rong
Software Institute
Nanjing University
Nanjing,Jiangsu,P.R.China
Email: ronggp@nju.edu.cn

Shenghui Gu
Software Institute
Nanjing University
Nanjing,Jiangsu,P.R.China
Email: SamuelGarciaSTK@gmail.com

He Zhang
Software Institute
Nanjing University
Nanjing,Jiangsu,P.R.China
Email: hezhang@nju.edu.cn

Dong Shao
Software Institute
Nanjing University
Nanjing,Jiangsu,P.R.China
Email: dongshao@nju.edu.cn

***Abstract*—As an emerging approach to support fast delivery of software features with reliable quality, DevOps attracts more and more practitioners and shows the potential to become one of the mainstream approach for software development and operation. Many universities begin to offer DevOps related courses to the students majored in software engineering and computer science. However, as a critical part of a DevOps course, the project practicing using DevOps might cast big challenges for teachers, compared to traditional project practicing. For example, the more frequent than ever delivery in DevOps practicing will inevitably increase the workload vastly for teachers to conduct effective evaluation. In this paper, we introduce a web based system (*DevOpsEnvy*) to support the management and monitoring of student teams practicing DevOps. By integrating several popular open source tools, this system provides students with features such as group management, project status monitoring and student performance data analysis, etc. Meanwhile, *DevOpsEnvy* system also provides teachers with sufficient evidence to perform evaluation. Our preliminary trial in Nanjing University revealed several advantages of *DevOpsEnvy* system.**

***Keywords*—DevOps, Education Support System, Project Management, Software Engineering**

I. Introduction

DevOps is emerging as a philosophy shift towards evolving software at a continuous pace and connecting each phase of the software lifecycle seamlessly. It aims at bridging the gap between the development team and the operation team as well as ensuring software quality. When talking about DevOps culture, automation plays an essential role in it since a high degree of automation is the cornerstone of quality deliveries with short cycle time [1] and automation also acts as a key to obtain rapid feedback [2].

In practice, various tools actually reflect automation to some degree. Furthermore, the pipeline which consists of various DevOps tools is the symbol of DevOps in particular. Currently, DevOps is crossing into more and more organizations under the leading of numerous well-known corporations like Facebook, Yahoo, Netflix, and Flickr [3]–[6]. Thanks to this, a lot of continuous delivery or deployment pipelines have been proposed by enterprises one after another, e.g., IBM, AWS and Microsoft [7]–[9]. The pipelines and tools facilitate the continuous software delivery, accelerate the resolution of problems, moreover, satisfy the customers under the pressure of ever-changing requirements.

Due to many advantages of DevOps, there is a demand for bringing DevOps into classroom for the purpose of making students understand practical DevOps and aware of its promising characteristics, hence better prepare them to modern software industry. Although there are several courses available on the Internet [10], [11], the topics are normally about how to use existing DevOps tools rather than helping students experience the DevOps methodology comprehensively. Several approaches have been proposed for teaching DevOps, unfortunately, these approaches usually remain at the theoretical stage and moreover, there are few specific software systems supporting the education and training of DevOps [12], [13].

There are many challenges hindering the progress of DevOps education to some extent. Two categories of these challenges have been identified, i.e. the skills acquisition and the technical environment [14]. Students need to be skillful and the environment has to be provisioned along with the proper functionality for evaluation to students' performance when it comes to teaching DevOps. Additionally, it is also not easy for students to work in teams without tool support. Therefore, it raises a strong demand to design a tool for providing a DevOps environment and an automatic evaluation approach as well from the perspective of teachers.

In this paper, we propose an education support system named *DevOpsEnvy* which utilizes several DevOps tools in order to help students perform DevOps practices easily and effectively. Students can collaborate with each other and manage there practicing projects conveniently via our system. Furthermore, our system could help teachers evaluate the work performance of student teams automatically, which dramatically reduces the workload of teachers.

The remainder of this paper is structured as follows. Section II reviews the related work in brief. Section III elaborates the design and the features of *DevOpsEnvy* system. In section IV, we developed an application demo and illustrated the application of *DevOpsEnvy* system. Section V discusses the potential issues when our system is adopted in other educational scenarios. The conclusions are drawn in Section VI with three possible improvements to *DevOpsEnvy* system as future work.

II. Related Work

A. Tools in Software Engineering Education

Project practicing plays a vital role in software engineering education, which has been recognized by many researchers and practitioners. The main purpose of project practicing is to help students apply theory to practices via a realistic (or near realistic) software project context. Generally, project practicing can be divided into two categories, i.e. the technology-specific project practicing [15], [16] and the process-specific project practicing [17]–[20], respectively. There are several tools supporting the evaluation of project practicing in both categories.

Raza et al. proposed a novel tool *ProcessPAIR* to support conducting the performance analysis of software developers [21]. Their tool can automatically identifies and ranks potential performance problems and root causes of individual developers according to a performance model calibrated from the performance data of many developers. Besides *ProcessPAIR*, there are also other tools existing for the sake of process education, for example, personal software process (PSP) [22], [23] and team software process (TSP) [24].

B. DevOps Tools

With the culture of DevOps deepened into practitioners, various types of DevOps tools have been springing up recently. As for continuous integration (aka, CI), there are several tools such as *Jenkins*, *TeamCity* and *Travis CI*[1], etc. Tools like *Ansible*, *Chef* and *Puppet* are specific for configuration management. When talking about container, *Docker* is one of the most popular systems. Besides, *Git*, *Subversion* and *Team Foundation Server (TFS)*[2] are used as version control systems. All of these types of tools can be categorized into DevOps tools in some sense.

Nevertheless, the increasing popularity of DevOps has led to an explosion of new tools focusing on automation, monitoring, testing, etc. XebiaLabs[3], a famous organization aiming to explore new frontiers in IT and then creating solutions for customers, proposes a periodic table of DevOps tools, in which totally 120 related tools with 15 types are involved. In this table, there are 12 CI tools, such as *Jenkins*, *Bamboo*, *Travis CI*, *TeamCity* and so on; 14 tools for deployment are listed, e.g., *Juju*[4] and *CodeDeploy*[5].

The explosion of DevOps tools makes it a big challenge for students, teachers even professionals in industry to choose suitable tool chain for their work. For example, they have to understand the differences among several tools which belong to the same type, e.g., *Jenkins* and *TeamCity*. Even though they have selected a set of tools, they still need to spend time learning how to coordinate these tools to finish a specific task. In an educational circumstance, it is really time consuming for both teachers and students. For this reason, we intend to integrate several DevOps tools into one system and offer the management functions in an effective and efficient manner.

C. Challenges in DevOps Education

DevOps is becoming a trend among software practitioners and it may be the dominating approach for software development and operation in the future. Nevertheless, most of the traditional software engineering courses only emphasize on the early and middle phases of the software lifecycle (i.e. requirements, design, programming, testing and tooling) while the production stage (i.e. deployment, maintenance) are often ignored or just treated in theory. In short, they teach "Dev" rather than "Ops" [14].

Therefore, more and more teachers want to devote themselves to bring DevOps into the classroom but there remains a number of challenges in the teachers' perspective. Christensen identified five challenges: a) teachers' experience; b) hybrid skills; c) emphasize skills; d) realistic environment and d) assessment and marking [14]. Furthermore, Christensen classified those challenges into two categories, skills acquisition and technical environment, except the teachers' experience. To be specific, skills acquisition means DevOps courses place a greater demand on students' skill rather than knowledge. Technical environment implies that a powerful platform is needed and an approach for proper and effective evaluation on students' performance is necessary.

With the purpose to address the aforementioned difficulties, several researchers proposed specific approaches for educational purpose. In the following paragraphs, we describe their work in brief.

Ohtsuki et al. proposed an education system *ALECSS* utilizing several DevOps tools in order to improve software quality and to provide quick feedback for both students and teachers [25]. *ALECSS* uses *Git* as its version control system and *Jenkins* for continuous integration. It utilizes *Ant* to build the source code. As for testing, it integrates *Findbugs*, *Checkstyle* and *JUnit* for static code analysis, coding style check and unit test respectively. The main feature of this system is to generate a report automatically for both teachers and students in order to perform evaluation. In addition, they performed a preliminary evaluation of *ALECSS* by comparing the warnings generated by tools with the student review reports. Nevertheless, *ALECSS* only deals with the challenge of assessment via bug checking.

[1]http://tinyurl.com/y8og8jlv
[2]http://tinyurl.com/y9xcc59g
[3]http://tinyurl.com/p3emhuz
[4]http://tinyurl.com/ngs5ec7
[5]http://tinyurl.com/lw3sopl

Eddy et al. introduced *CDEP* (Continuous Delivery Educational Pipeline) to solve the problems faced by teachers when teaching continuous integration and continuous delivery in software engineering courses [26]. Similar to *ALECSS*, *CDEP* integrates *Jenkins*, *Git*, *Docker* and *MySQL* to implement the pipeline. The main goal of *CDEP* is to provide a portable, reusable, visual and quick to setup teaching tool. If the teachers are not familiar with the plethora of complex tools, *CDEP* would give them the ability to focus more on the material and not the complexities of provisioning the entire pipeline. As a matter of fact, *CDEP* only solves the problem of realistic environment via a pipeline.

However, there still exists the necessity to have a system which is able to deal with both categories of challenges (i.e. skills acquisition and technical environment) at the same time. When it comes to DevOps education tools, there ought to be a full-scale, easy-to-use and standardized tool or system in order to provide a out-of-the-box DevOps environment and an automatic evaluation approach.

III. System Design

A. Overview

As a web-based system, *DevOpsEnvy* integrates several prevalent open source tools, based on which, several features are designed and implemented to support educational purpose. Students are able to practice DevOps by developing software projects based on *DevOpsEnvy* system. Basically, students only need to focus on the development of software artifacts rather than configuration of the development environment. *DevOpsEnvy* system consists a concise set of tools. Meanwhile, several key parameters (e.g., the pipeline execution stages, test parameters, etc.) can be modified via scripts edited by students. In the meantime, teachers are able to monitor the status of each student project as well as the performance of each student. At the end of a normal semester, these status data are automatically collected for teachers to evaluate the whole practicing process and provide basis for grading students as well. *DevOpsEnvy* system can reduce the workload for teachers to perform manual evaluation, which is especially useful in a DevOps course.

Various tools are integrated into *DevOpsEnvy*. We use *Git* as the version control system and *GitHub* as the web-based repository for *Git*. *Jenkins*[6] provides the ability of continuous deployment for our system. Another imperative DevOps tool is *SonarQube*[7], which makes continuous inspection possible. *Maven* and *Gradle* are mainly used to build the source code and generate the artifacts so as to deploy into containers. *Docker*[8] is provisioned in advance for students to conduct continuous deployment.

In the following sections, we elaborate the workflow of DevOps practice as well as the architecture and features of *DevOpsEnvy*.

[6]http://tinyurl.com/zy22rk7

[7]http://tinyurl.com/y7lsunpp

[8]http://tinyurl.com/o4vvrlw

B. Workflow of DevOps Practice

DevOpsEnvy is a teaching support system which collects and analyzes the status data, and then provides feedback for both students and teachers. The status data are collected from the process of developing software projects. Nevertheless, different teams may apply different workflows and the DevOps tools they used may vary. Hence, a workflow for DevOps practicing needs to be predefined in our system. Otherwise, teachers will find it difficult to monitor and evaluate the performance of student teams consistently. As a prerequisite for *DevOpsEnvy*, the workflow we defined is a loop pipeline for the purpose of helping developers to improve the code quality continuously. The steps of the workflow are outlined in the following and the entire workflow is illustrated in Fig. 1.

Step 1: Students develop source code in their integrated development environment or text editor according to their preferences.

Step 2: After coding, students commit their changes via *Git*, and then push them into *GitHub*.

Other version control systems, also known as source code management tools, can be applied to manage the changes of source code as substitutions for *Git*, for example, Subversion and TFS Version Control [12]. Our system does not specify the version control system and students can use whatever they prefer as long as *Jenkins* supports them. The reason to choose *Git* and *GitHub* is that *Git* is an open source system as well as one of the most popular version control tools. Moreover, *GitHub* offers free repositories on the clouds, therefore, students do not need to setup the repositories locally. Meanwhile, *GitHub* allows separated developers to collaborate on the same project seamlessly and easily.

Step 3: Once the source code is updated, *Jenkins* automatically starts the pipeline by the specific trigger or timer defined by students.

Jenkins is the leading open source automation server, aiming at supporting the automatic build and deployment of the software artifacts, which makes continuous integration or continuous deployment possible [1], [27], [28]. Since *Jenkins 2.0*, code pipeline is supported and it is a powerful feature which allows developers to do define many tasks, so we select *Jenkins* as the continuous deployment server. The behavior of *Jenkins* pipeline is controlled by *Jenkinsfile* which is a groovy script. Students can modify the script if they want more stages when execute the pipeline. Usually, the pipeline script defines four stages: a) Pull the project source code into *Jenkins* workspace from *GitHub*; b) Perform tests via *SonarQube*; c) Build the source code and generate the corresponding artifacts via specific build tools; d) Deploy the artifacts packaged into *Docker* containers.

Step 4: *SonarQube* performs a series of tests and outputs the analysis results.

SonarQube is an open source code quality management platform. It provides the capability to not only show health of an application but also to highlight issues newly introduced. The greatest strength of using *SonarQube* is that it integrates

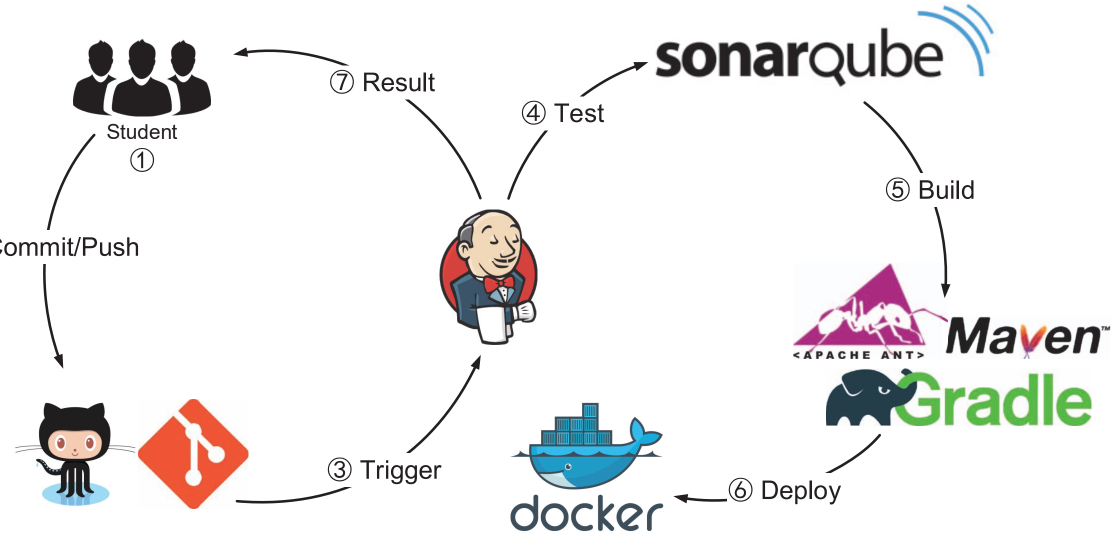


Fig. 1. Workflow of DevOps Practice

several wide used testing tools such as *Findbugs*[9], *Checkstyle*[10] and *JUnit* into one system in the form of plugins. Not merely presenting the results from each plugin, *SonarQube* reprocesses these results via unique algorithms and measures them quantitatively in the end. Not only static tests but also dynamic tests can be conducted through *SonarQube*. That is why we utilize *SonarQube* to facilitate testing.

In our system, *SonarQube* mainly performs three kinds of tests (i.e. static test, unit test and coverage test) in sequence.

- *Static Test: SonarQube* utilizes several static test tools to analyze the source code and discover the code smells, bugs and vulnerabilities. In addition, there are other languages supported apart from Java in *SonarQube*. Teachers can choose various kinds of plugins needed in their courses from the SonarQube plugin library.
- *Unit Test:* Students ought to write the test code and *SonarQube* will execute the test cases, finally, generates a report containing the results of tests.
- *Coverage Test:* Plugins such as *JaCoCo*[11] and *Cobertura*[12] are able to generate the code coverage reports which can assess whether the test cases are sufficient.

All of the test reports are synthesized and our system selects the necessary data which help the improvement of the code quality and the evaluation of the projects.

Step 5: After testing, *Jenkins* builds the source code and generates artifacts via specific build tools.

The build tool is determined by the project. Either Ant, Maven or Gradle is available for building. The build status can be obtained from *Jenkins* and it will be shown in our system.

Step 6: If the artifact is generated successfully, *Jenkins* will put it into a *docker* container and run the container automatically. If the project is a web-based system, then the URL of the project is available in our system.

Compared with virtual machine, container has a better performance when it comes to continuous deployment and *Docker* is one of the most famous open source container engine [29], [30]. The commands can be defined in *Jenkinsfile* for the purpose of deploying artifacts into containers.

Step 7: Students ought to modify their code based on the feedback from *DevOpsEnvy* and start a new cycle from step one.

The workflow is designed to help students experience the advantages of DevOps development when it comes to the rapid and unpredictable changes of requirements raised by teachers. When using traditional development mode, developers may not deliver the products on time under the pressure of ever-changing demands. In addition, bugs may not be fixed and then accumulated, ending up with an awful software product. Compared with the traditional development, DevOps development is able to make up for the drawbacks, giving the ability of continuous deployment and ensuring the code quality.

Based on the workflow, we developed *DevOpsEnvy* system to obtain status data in some important process, e.g., both the build phase and test phase. Our system not only provides a DevOps environment for process-specific project practicing but also the process data for monitoring and evaluation.

C. Architecture

DevOpsEnvy is a web-based system and adopts a three-tier architecture. Fig. 2 shows the details of the architecture.

[9]http://tinyurl.com/346lac
[10]http://tinyurl.com/8k4w
[11]http://tinyurl.com/brhqy32
[12]http://tinyurl.com/lhpstsk

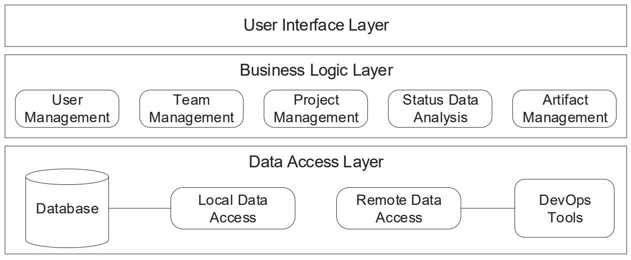


Fig. 2. Architecture of *DevOpsEnvy* System

User Interface Layer is the entrance of *DevOpsEnvy*. The operations from end users should be handled in this layer.

The middle of three layers is business logic layer which deals with the raw data and transmit them to user interface layer for presentation. This layer contains the logic and algorithms of our system. For example, the logic for parsing the raw data obtained from *Jenkins* and the algorithm for calculating the build frequency. The functions can be classified into five features: user management, team management, project management, status data analysis and artifact management. The details of these features will be elaborated in section III-D.

As for data access layer, the main task is to obtain information data from both local and remote tools and severs. The raw data from web are primarily captured from *Jenkins* and *SonarQube*. Fortunately, *SonarQube* offers a full set of web APIs for developers to use. However, the APIs of *Jenkins* are not full-scale and consolidated and then we have to design and implement corresponding modules. On the other hand, the raw data stored locally are the information generated by our *DevOpsEnvy* system.

D. Feature Of DevOpsEnvy

DevOpsEnvy provides imperative environment for practicing DevOps according to the workflow defined in section III-B. Students do not need to select and configure various DevOps tools since each necessary DevOps tool is configured by *DevOpsEnvy* in advance. *DevOpsEnvy* itself is enough to perform necessary operations when it comes to practicing DevOps by students. Furthermore, the evaluation of projects and student teams as well is much more easier and feasible via *DevOpsEnvy*. The following paragraphs describe the features in detail.

1) User Management: Manage the user information and synchronize it with other DevOps tools.

It is impractical for students to manage their user information in each integrated DevOps tools. For instance, without an integrating system, students have to register in several different systems such as *Jenkins* and *SonarQube* separately, which obviously block the data flow from one tool to another. In addition, from the teachers' perspective, various tools selected by different student teams may also increase the difficulty in performance evaluation to student teams.

Using *DevOpsEnvy*, students only need to register their information in *Jenkins* and other tools such as *SonarQube* could share the same user information. *DevOpsEnvy* also use these information to manage users.

2) Team Management: Manage team members for better collaboration.

As we know, job contained in one project is the minimum unit in *Jenkins* and nearly all operations are job oriented. Although *Jenkins* has features such as user management, jobs are not automatically associated with users. In this sense, there is no team management in *Jenkins*, which brings certain troubles when developers want to work in a team. They may require a permission management function that the source code is only available among team members. Besides, they may want to know how they contribute to the team but there is no suitable metric in *Jenkins*. In educational environment, team management and performance evaluation are necessary.

DevOpsEnvy provides the team management function so as to enhance the *Jenkins* tool to meet specific requirements in educational environment. Multiple team members are permitted to be included in one team by one student. Meanwhile, all the students are able to quit any team they have already joined in as they wish. For security purpose, students cannot access the detailed information of projects which they do not participate in.

3) Project Management: Manage the DevOps projects by creation, modification and removal of project information within one step.

Initializing and provisioning a project in *Jenkins* may be not easy for some students. Failures in this step perhaps discourages the students' enthusiasm to using the tool. Moreover, there are a lot of types of jobs and students may use improper types, which increases the workload of checking. In addition, modifying and removing projects can be cumbersome because each DevOps system needs to be operated manually at the same time without an integrated system such as *DevOpsEnvy*.

In order to simplify the process to initialize a new project, our *DevOpsEnvy* system integrate the project creation and provision process in one system. With the aid of *DevOpsEnvy*, students can easily create a practicing project with minimal information such as project name. Once this information is given, *DevOpsEnvy* will initialize the pipeline project in *Jenkins* utilizing the job template file *config.xml*. The behavior of the pipeline is defined in *Jenkinsfile* by students. For example, the project source code will be tested by *SonarQube* after it is pulled from *GitHub*, and then *Jenkins* will build an artifact for *Docker* to deploy. Besides, the project configuration is also predefined in *SonarQube*. The modification and removal of project information are also handled by *DevOpsEnvy*.

4) Performance Data Analysis: Present the performance status for teams and individuals in practicing projects.

Though there are numerous metrics provided by *SonarQube*, not every one of these metrics is valuable for the purpose of evaluating the project and student performance in this project as well. As for *Jenkins*, it lacks necessary data to create comprehensive report for users to monitor the progress and status of DevOps projects.

To address these issues, *DevOpsEnvy* provides the status data analysis function for the purpose of generating useful reports. This function is divided into two parts, i.e. team analysis and individual analysis, respectively. The results of team analysis reflect the status of the whole project, e.g., the total number of violations (to coding standards, for example) and the test results, which are valuable to help student teams to understand and improve the quality of source code. Additionally, the project evolution status data are analyzed especially for *Jenkins* in order to help teachers to monitor the process of projects and evaluate the performance of each team. For instance, the build frequency in the last ten builds and the successful rate in the last week. On the other hand, individual analysis presents the filtered information based on user names, for example, all the issues in a certain project introduced by a certain team member. It is helpful for students to assign their responsibilities for development and maintenance of the DevOps project. Meanwhile, it may also encourage a self-managed culture.

5) Artifact Management: Manage and check artifacts after deployed.

DevOpsEnvy utilizes *Docker* for artifact deployment. This is the final step of the total pipeline. According to the script in *Jenkinsfile*, the packaged artifacts will be deployed into a *Docker* container. However, when the practicing projects are web-based systems, the entry URLs of projects will be distinct. Manual checking may not be feasible, given the heavy workload to looking up and checking all the URLs.

Hence, *DevOpsEnvy* shows the URLs of artifacts on one page for easy access. When it comes to evaluating, *DevOpsEnvy* can test the accessibility of the artifacts for teachers.

E. Metrics Applied in DevOpsEnvy

Metrics represent the status of the practicing projects and the developers in a quantitative manner. Students can be aware of the aspects in which they need to improve the performance while teachers can evaluate the projects through a set of metrics. Both of them are able to take the advantage of the feedback based on the metrics more or less. In the following, we describe these useful metrics in detail.

- **Build Frequency** is a mean duration between two adjacent builds. It represents the performance of continuous deployment conducted by each team.
- **Build Success Rate** is calculated according to the build result at each time point. It reflects the health of projects.
- **Build Condition** records the detailed information of each build including the build result, build time and duration.
- **Complexity** is calculated based on the number of paths through the code. The complexity gets increased by one whenever the control flow of a function splits. For instance, if there is one "if" statement occurs in a function, then the complexity of this function will be added by one. This metric represents the degree of relationship in a system. The larger complexity is, the more intricate the code is and the worse the project might be.
- **Lines Of Code** represents the physical lines that contain at least one character which is neither a whitespace or a tabulation or part of a comment. This metric reflects the workload of the project.
- **Comments(%)** represents the density of lines containing either comment or commented-out code.

  $$Comments(\%) = \frac{Comment\ lines * 100}{(Lines\ of\ code + Comment\ lines)}$$

  The density of comments reflects the readability of the source code.
- **Duplicated Lines(%)** represents the density of duplicated lines.

  $$Duplicated\ Lines(\%) = \frac{Duplicated\ lines * 100}{Lines}$$

  There might be some code smells if the density of duplication is large.
- **Technical Debt** represents the effort to fix all maintainability issues. Developers might spend more time on fixing issues if the duration of technical debt is longer.
- **Severity Of Issues** contains five levels, i.e. *Blocker*, *Critical*, *Major*, *Minor* and *Info*. The severity is descending and each one informs the number of issues at this level.
- **Type Of Sssues** contains three levels, i.e. *Code smell*, *Bug* and *Vulnerability*. Code smell refers to any symptom in the source code of a program that possibly indicates a deeper problem. A bug is when a system is not behaving as it is designed to behave. A vulnerability is a way of abusing the system whether that is due to a design fault or an implementation fault.
- **Quality Gate** is the analysis status associated to the project. If it is "Ok", then your project has passed the tests predefined by the teacher.

Not every metric is useful for each condition, so teachers could choose and define suitable metrics to evaluate the projects in a specific situation. For example, teachers could only monitor the **Build Frequency** at the middle of the practicing project to understand how the students develop source code.

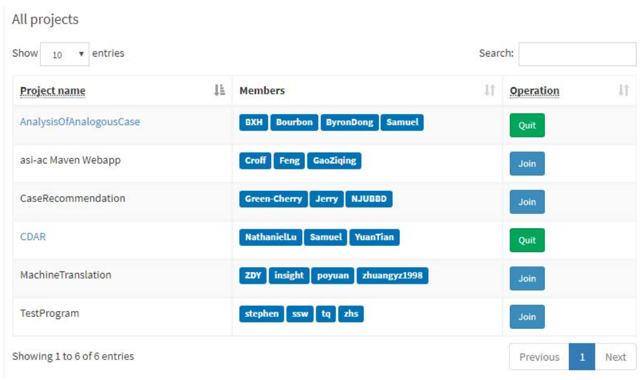
IV. Application Demo

In order to demonstrate our system in a clear manner, we follow the process from project creation to artifact deployment.

A. Preparation and Project Creation

In the first place, every student needs an account to use the functions in *DevOpsEnvy*. The administrator of *DevOpsEnvy* system (usually teachers or teaching assistants) could help to set up accounts for all the students in a DevOps practicing course. The user interface after login is shown as Fig. 3.

If student team wants to use *GitHub* to manage their source code during the practicing process, they should create projects in *GitHub* first and input the URL of the project repository (in *GitHub*) in a page such as Fig. 4. And then they should edit the pipeline script *Jenkinsfile* which defines the execution stages in *Jenkins* as well as the configuration file


All projects

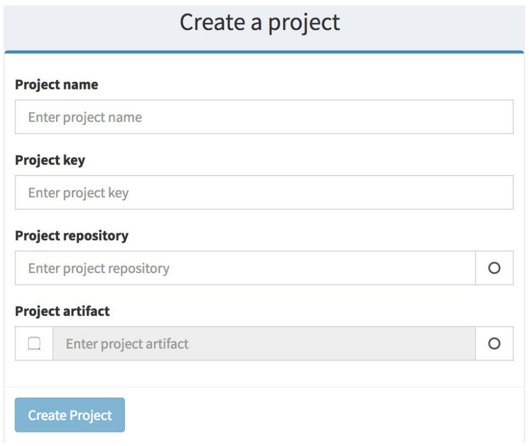
Show 10 entries Search:

Project name	Members	Operation
AnalysisOfAnalogousCase	BXH Bourbon ByronDong Samuel	Quit
asi-ac Maven Webapp	Croff Feng GaoZiqing	Join
CaseRecommendation	Green-Cherry Jerry NJUBBD	Join
CDAR	NathanielLu Samuel YuanTian	Quit
MachineTranslation	ZDY insight poyuan zhuangyz1998	Join
TestProgram	stephen ssw tq zhs	Join

Showing 1 to 6 of 6 entries

Fig. 3. Project List

sonar-project.properties for *SonarQube* analysis. These files are unique in different projects and they should be placed in the root directory of the whole project. Next, they can initialize projects via *DevOpsEnvy* system after they login. Fig. 4 shows the user interface for creating a project.


Create a project

Project name
Enter project name

Project key
Enter project key

Project repository
Enter project repository

Project artifact
Enter project artifact

Fig. 4. Project Creation

Name repetition is allowed for the project name while the project key must be unique because the project key is an ID in *SonarQube* and it is required when sending requests via build-in APIs. The project repository is the URL of your project repository recorded in advance. If the project was built before and already has been deployed, then the project artifact field can be filled in with the URL of the artifact. In this procedure, *DevOpsEnvy* will create a project both in *Jenkins* and *SonarQube* using the information given by students. Meanwhile, the system initializes and provisions the project in *Jenkins* automatically. In the end, these information will be stored in *DevOpsEnvy*.

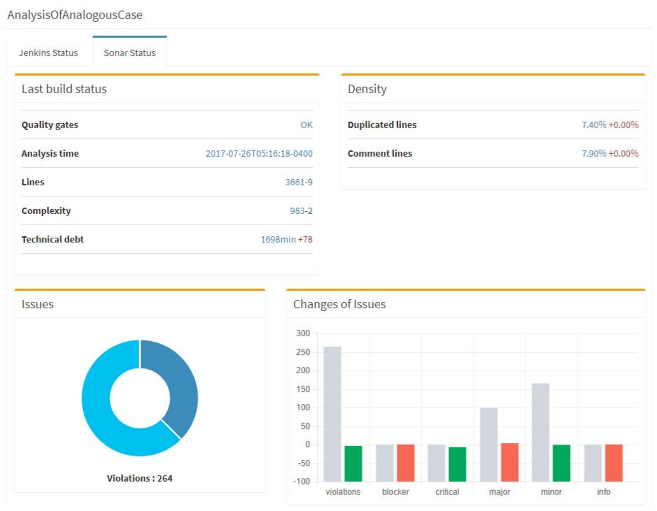
The created projects will be shown in the project list (shown as Fig. 3). This list displays some essential information about a specific project, e.g., project name and project members, and the detailed information is also available when clicking the project name.

B. Team Management

Joining in and quitting the project teams can be simply conducted via the button in "Operation" column. The "Join" button works when a student was not a team member for a certain project and vice verse, the "Quit" button. Moreover, one student can take part in several different project teams.

C. Testing

Till this step, the creation and initialization of practicing projects are finished. After creating projects, students should develop projects and try to satisfy the ever-changing requirements raised by their teachers. The project source code ought to be updated frequently, for instance, one commit per day and even more, three commits per day. Each time the source code changes, the *Jenkins* pipeline is triggered by the hook in *GitHub*. The source code will be analyzed in *SonarQube* after pulled into the workspace locally. The analysis results will be stored in *SonarQube* database and a set of metrics we collected for improvement and evaluation will be displayed in our *DevOpsEnvy* system. Fig. 5 shows some analysis results.


AnalysisOfAnalogousCase

Last build status

Quality gates	OK
Analysis time	2017-07-26T05:16:18-0400
Lines	3661-9
Complexity	983-2
Technical debt	1698min +78

Density

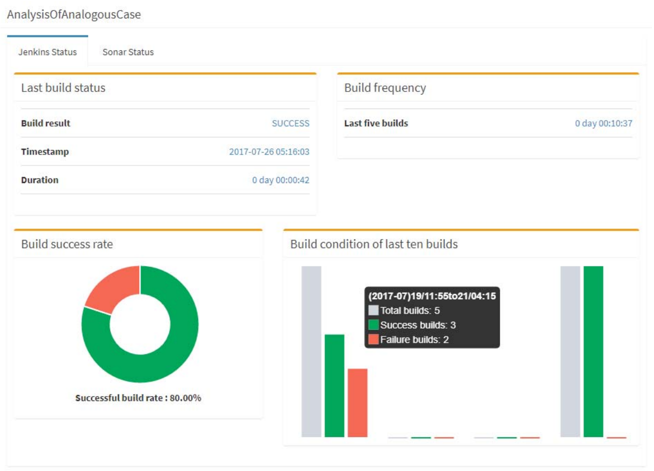
Duplicated lines	7.40% +0.00%
Comment lines	7.90% +0.00%

Fig. 5. Test Results

The metrics displayed on the interface are elaborated in section III-E. The numbers with sign represent the changes since the last analysis. The positive sign and the negative sign mean the increasing and the decreasing of the numbers respectively. In addition, there are two distinct types of color–red and green in this analysis result page. Color red represents the improvement while green means the deterioration. We utilize doughnut chart to illustrate the percentage of each level of issues. In order to make the results more distinct, we also color the issue according to the severity, e.g., red for "Blocker" issues, green for "Info" issues and blue for the middle level. Furthermore, the number of changed issues grouped by severity level is shown in the form of a bar chart comparing with the sum number correspondingly.

D. Building

The stage after testing is to build. *Jenkins* will package the source code via a specific build tool (e.g., *Maven*) and generate an artifact (e.g., WAR file). The build results are displayed in *DevOpsEnvy* as well (Fig. 6). The last build status is given and the recent **Build Frequency** is also displayed. Our system calculates the ratio of the successful builds and groups the condition of each build by the time interval. These results are valuable especially for teachers to monitor and evaluate the performance of each team.

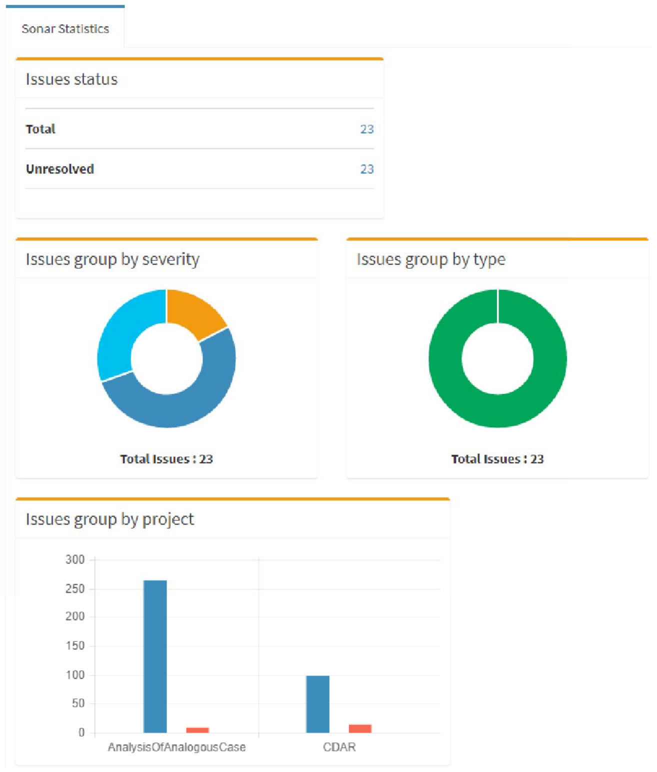


Build result	SUCCESS
Timestamp	2017-07-26 05:16:03
Duration	0 day 00:00:42
Last five builds	0 day 00:10:37

Successful build rate : 80.00%

(2017-07)19/11:55to21/04:15
Total builds: 5
Success builds: 3
Failure builds: 2

Fig. 6. Build Results



Total	23
Unresolved	23

Total Issues : 23

Total Issues : 23

Fig. 7. User Status

Apart from the team results, *DevOpsEnvy* system also provides the performance status information for each developers. As shown in Fig. 7, issues introduced by users are grouped by severity and projects they take part in. Various types of color and charts are used for an explicit presentation.

E. Deployment

Finally, if the artifact is generated without failure, *Jenkins* will deploy it into a *Docker* container according to the commands defined in *Jenkinsfile* automatically. The URL of the artifact is shown in the basic information area located in the right of the web page for easy access.

The aforementioned process is the first workflow when practicing DevOps. Students do not need to seek for proper DevOps tools henceforth and they can spare more time to develop and improve their code continuously according to the analysis results. At the end of the practicing, teachers can evaluate the performance of project teams based on the information in a relaxing way thanks to the relatively light workload required by *DevOpsEnvy*.

F. Case in Nanjing University

We applied this system in Nanjing University. Fig. 3 shows the actual projects that the students created during the practice session of our DevOps course. Owing to *DevOpsEnvy* system, students did not bother to provision the development environment in their own devices. Besides, it also reduced the problems that students encountered and saved the time of teachers or teaching assistants to address these problems.

In this course, *DevOpsEnvy* system was proved to be helpful in collecting process data. Fig. 5 and Fig. 6 shows the build results and test results of one real project developed by a student team. The process data were accessible when the students performing development activities. At the end of the course, these data were used to evaluate the performance of the student teams. In general, *DevOpsEnvy* meets our purposes that simplifying the development environment and make the process data easily accessible. *DevOpsEnvy* system provided them with an easy-to-use development environment to practice DevOps, allowing students to build their code frequently and ensuring the code quality.

V. Discussion

The main contribution of this paper is that we propose a web-based system to support the management and monitoring of student teams practicing DevOps for educational purpose. Students can use *DevOpsEnvy* to collaborate with others and improve the code quality according to the analysis results. Teachers could monitor the work status and evaluate the performance of each student team via this system. As an attempt at this preliminary stage, *DevOpsEnvy* system meets our exception to carry out a DevOps course and practicing DevOps. Nevertheless, we also identified several issues which need to be discussed here for the considerations when utilizing this system in other educational scenarios.

A. Customizability

At this stage, *DevOpsEnvy* mainly integrates *Jenkins*, *SonarQube* and *Docker* along with version control systems and build tools. We applied a hard-coded strategy to simplify development complexity and provide the system for students as fast as possible. This strategy inevitably impacts the customizability of *DevOpsEnvy* system. In fact, when applying our system into other educational scenarios, the different build-in tools might not be suitable. It is better to let teachers customize and define the DevOps tools to meet their specific purposes. In our near future expectation, *DevOpsEnvy* could blossom out into a plugin-based framework which allows teachers to customize the DevOps tools to adapt the de facto condition in a comfortable way.

B. Metrics Definition

Currently, *DevOpsEnvy* provides several metrics for both teachers and students to evaluate the work status. However, the set of metrics for evaluation is based on the data available in existing tools, which has not been defined with a broad consideration. With increased experience and understanding of DevOps education, we might need more metrics from various perspectives. In this sense, *DevOpsEnvy* need to be enhanced to support defining more metrics. Besides, an automatic tool integrating different algorithms could be developed to generate the grades for students in diverse perspectives, for example, productivity, quality and contribution, etc.

C. Automatic Evaluation Tool

Most of the functions in *DevOpsEnvy* are designed particularly for students to practice DevOps, such as team management and project management. Though the work status is analyzed and provided by our system, the evaluation is still a semi-automatic process rather than complete automation. In our application demo, teachers need to access the information for each student to evaluate the work performance respectively. It indeed reduces the workload of evaluation in some degree compared with manual checking entirely. Nevertheless, it is still not a good approach when it comes to large numbers of student teams. A support tool have to be proposed to perform full-automatic evaluation. Teachers could customize the evaluation criteria to cope with different situations and then a comprehensive report could generated automatically.

VI. Conclusion

Due to the lack of supporting tools, the DevOps education becomes much more difficult for teachers to conduct. This paper proposes an education support system utilizing different DevOps tools in order to aid both students and teacher when it comes to DevOps practicing. The collaboration of students is simplified while the workload of evaluation is reduced for teachers. Our preliminary trial in Nanjing University received very positive results. *DevOpsEnvy* not only help students focus on the DevOps method instead of tools and technologies, but also facilitate teachers to conduct performance evaluation automatically, which reduced the teachers' workload greatly.

In the future, we will perfect our system continuously so as to adjust to a more intricate environment. First of all, a standard for evaluation via metrics should be proposed and then validated, hence support a full-automatic evaluation to both projects and students. Second, we should also provide features to support customization on both tools and metrics.

References

[1] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Software*, vol. 33, no. 3, pp. 94–100, May 2016.

[2] M. Hüttermann, *DevOps for Developers*, ser. Expert's Voice in Web Development. New York: Apress, 11 September 2012.

[3] L. E. Lwakatare, P. Kuvaja, and M. Oivo, "Dimensions of DevOps," in *Proceedings of the 16th International Conference on Agile Software Development (XP '15)*. Cham: Springer, 25 May 2015, pp. 212–217.

[4] F. Erich, C. Amrit, and M. Daneva, "A mapping study on cooperation between information system development and operations," in *Proceedings of the 15th International Conference on Product-Focused Software Process Improvement (PROFES '14)*. Cham: Springer International Publishing, 10 December 2014, pp. 277–280.

[5] S. K. Bang, S. Chung, Y. Choh, and M. Dupuis, "A grounded theory analysis of modern Web applications: Knowledge, skills, and abilities for DevOps," in *Proceedings of the 2nd Annual Conference on Research in Information Technology (RIIT '13)*. New York, NY, USA: ACM Press, 10 October 2013, pp. 61–62.

[6] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and deployment at Facebook," *IEEE Internet Computing*, vol. 17, no. 4, pp. 8–17, July 2013.

[7] IBM, "Delivery pipeline." [Online]. Available: https://console.ng.bluemix.net/catalog/services/delivery-pipeline

[8] AWS, "AWS codePipeline." [Online]. Available: https://aws.amazon.com/codepipeline/

[9] Microsoft Azure, "Visual Studio Team Services." [Online]. Available: https://azure.microsoft.com/en-us/services/visual-studio-team-services/

[10] DevOpsGuys, "Educate - UK DevOps training courses and programmes." [Online]. Available: https://www.devopsguys.com/devops-educate/

[11] DevOps Institute, "Setting the standard in DevOps training." [Online]. Available: http://devopsinstitute.com/

[12] M. Airaj, "Enable cloud DevOps approach for industry and higher education," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 5, 10 March 2017.

[13] S. Krusche and L. Alperowitz, "Introduction of continuous delivery in multi-customer project courses," in *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion '14)*. New York, NY, USA: ACM, 31 May 2014, pp. 335–343.

[14] H. B. Christensen, "Teaching DevOps and cloud computing using a cognitive apprenticeship and story-telling approach," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. New York, NY, USA: ACM, 11 July 2016, pp. 174–179.

[15] J. C. Adams, "Chance-It: An object-oriented capstone project for CS-1," *SIGCSE Bull.*, vol. 30, no. 1, pp. 10–14, March 1998.

[16] J. O. Hamblen, H. L. Owen, S. Yalamanchili, and B. Dao, "An undergraduate computer engineering rapid systems prototyping design laboratory," *IEEE Transactions on Education*, vol. 42, no. 1, pp. 8–14, February 1999.

[17] D. A. Umphress, T. D. Hendrix, and J. H. Cross, "Software process in the classroom: The capstone project experience," *IEEE Software*, vol. 19, no. 5, pp. 78–81, September 2002.

[18] D. P. Groth and E. L. Robertson, "It's all about process: Project-oriented teaching of software engineering," in *Proceedings of the 14th Conference on Software Engineering Education and Training (CSEET '01)*. IEEE, 19 February 2001, pp. 7–17.

[19] B. R. von Konsky and M. Robey, "A case study: GQM and TSP in a software engineering capstone project," in *Proceedings of the 18th Conference on Software Engineering Education Training (CSEET '05)*. IEEE, 18 April 2005, pp. 215–222.

[20] S. Jarzabek and P.-K. Eng, “Teaching an advanced design, team-oriented software project course,” in *Proceedings of the 18th Conference on Software Engineering Education Training (CSEET ’05)*. IEEE, 18 April 2005, pp. 223–230.

[21] M. Raza, J. ao Pascoal Faria, and R. Salazar, “Helping software engineering students analyzing their performance data: Tool support in an educational environment,” in *Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C ’17)*. Piscataway, NJ, USA: IEEE Press, 20 May 2017, pp. 241–243.

[22] W. S. Humphrey, *PSP^{SM}: A Self-Improvement Process for Software Engineers*. Addison-Wesley Professional, 2005.

[23] G. Rong, H. Zhang, S. Qi, and D. Shao, “Can software engineering students program defect-free?: An educational approach,” in *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE ’16)*. New York, NY, USA: ACM, 14 May 2016, pp. 364–373.

[24] W. S. Humphrey, *Introduction to the Team Software Process*. Addison-Wesley Professional, 2000.

[25] M. Ohtsuki, K. Ohta, and T. Kakeshita, “Software engineer education support system ALECSS utilizing DevOps tools,” in *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services (iiWAS ’16)*. New York, NY, USA: ACM, 28 November 2016, pp. 209–213.

[26] B. P. Eddy, N. Wilde, N. A. Cooper, B. Mishra, V. S. Gamboa, K. N. Patel, and K. M. Shah, “CDEP: Continuous delivery educational pipeline,” in *Proceedings of the SouthEast Conference (ACM SE ’17)*. New York, NY, USA: ACM, 13 April 2017, pp. 55–62.

[27] V. Armenise, “Continuous delivery with Jenkins: Jenkins solutions to implement continuous delivery,” in *2015 IEEE/ACM 3rd International Workshop on Release Engineering (RELENG ’15)*. IEEE, 19 May 2015, pp. 24–27.

[28] M. de Bayser, L. G. Azevedo, and R. Cerqueira, “ResearchOps: The case for DevOps in scientific applications,” in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM ’15)*. IEEE, 11 May 2015, pp. 1398–1404.

[29] A. A. Mohallel, J. M. Bass, and A. Dehghantaha, “Experimenting with Docker: Linux container and BaseOS attack surfaces,” in *2016 International Conference on Information Society (i-Society ’16)*. IEEE, 10 October 2016, pp. 17–21.

[30] M. A. R, J. K. Patel, S. Akhtar, V. K. Agrawal, and K. N. B. S. Murthy, “Docker container security via heuristics-based multilateral security-conceptual and pragmatic study,” in *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT ’16)*. IEEE, 18 March 2016, pp. 1–14.