Research Article

Mohammad Zarour, Mohammed Akour, Mamdouh Alenezi*

Enhancing DevOps Engineering Education Through System-Based Learning Approach

https://doi.org/10.1515/edu-2024-0012
received February 02, 2024; accepted April 17, 2024

Abstract: System-based learning (SBL) in engineering domains integrates systems thinking and engineering principles to develop a system. In software engineering, to develop software using the DevOps process, using SBL environment, students gain a comprehensive understanding of the DevOps software development process and apply theoretical concepts to real-world problems by implementing a complete system pipeline, encompassing the design, development, testing, and deployment of software systems. This article introduces an SBL approach to teaching DevOps engineering, addressing the limitations of traditional methods in equipping students with the necessary skills and knowledge. To evaluate the effectiveness of the SBL approach, a case study was conducted to teach a DevOps course within an undergraduate software engineering program. Students completed a project involving the implementation of a system pipeline from requirement gathering to deployment. Results from the case study demonstrate that the SBL approach has improved students' understanding of DevOps engineering and the software development big picture. The approach enhanced students' systems thinking and problem-solving capabilities and prepared students for the challenges of a rapidly evolving technological landscape.

Keywords: DevOps engineering, system-based learning, software development process, systems thinking, project-based learning

1 Introduction

A system is a set of elements that are connected to each other by feedback relationships and organized in a way that achieves a function (Meadows, 2008). Because of the value added by the relationships between the various components, a system is more than the sum of its parts (Meadows, 2008). Higher education fields, including business, engineering, and medicine, demand that students be able to understand and operate complex systems. To deliver such diverse areas, universities and colleges require more than just traditional, isolated learning methodologies. A useful framework for comprehending the interdependence of the various components that make up a system and promoting a more comprehensive approach to learning and change is provided by system thinking. System thinking has fundamentally changed both how problems are conceptualized and how solutions are approached (Ndaruhutse, Jones, & Riggall, 2019).

Systems thinking approaches are different from what is known as deterministic approaches. Deterministic approaches concentrate on breaking down systems into their constituent elements. The underlying presumption is that the system as a whole can be comprehended if its constituent parts can (Silberstein & Spivack, 2023; Walker, Stanton, Salmon, Jenkins, & Rafferty, 2010). Since more than 50 years, deterministic approaches have advanced fields like safety science and human factors and ergonomics. However, deterministic approaches are not suitable for all fields; fields like artificial intelligence, and new advancements in information and communication technology require more system-based approaches (Ackoff, 1973; Lee & Rine, 2004; Sharma, 2006; Sterman, 2018). Moreover, deterministic approaches are found to be unsuitable for the fourth industrial revolution, the internet of things, ecological sustainability, climate change adaptation, and food and farming systems. Problems such as these are increasingly recognized as "systems problems" (Dul et al., 2012; Salmon, Walker, Read, Goode, & Stanton, 2017; Wilson, 2014), and applying reductionist methods to tackle such problems has several limitations that have been highlighted by various researchers, see, for example, Salmon et al. (2017), Walker, Salmon, Bedinger, & Stanton (2016), Walker et al. (2010), and Woods and Dekker (2000).

Hence, system thinking is a problem-solving approach that encourages looking beyond individual components and examines the bigger picture. It focuses on

* **Corresponding author: Mamdouh Alenezi,** Department of Software Engineering, Prince Sultan University, Riyadh, Saudi Arabia, e-mail: malenezi@psu.edu.sa
Mohammad Zarour: Department of Software Engineering, The Hashemite University, Zarqa, Jordan
Mohammed Akour: Department of Software Engineering, Prince Sultan University, Riyadh, Saudi Arabia

Open Access. © 2024 the author(s), published by De Gruyter. This work is licensed under the Creative Commons Attribution 4.0 International License.

understanding how parts of a system work together to achieve a whole, considering the feedback loops and unintended consequences of actions. In the context of the DevOps development process, the development system encompasses development and operation processes, tools, team members, and software users. By understanding how these elements interact, we can make more informed decisions about improving the overall DevOps experience.

However, the inherent complexity and interdisciplinary nature of DevOps engineering pose significant challenges for traditional pedagogical approaches, which often emphasize compartmentalized and theory-heavy learning (Ferino et al., 2023; Fernandes, Ferino, Kulesza, & Aranha, 2020; Jones, 2019a). Teaching DevOps engineering effectively requires students to understand and apply a range of complex concepts and tools, such as continuous integration/continuous delivery (CI/CD) pipelines, containerization, and infrastructure as code (IaC) (Fernandes et al., 2022; Jones, 2019b; Leite, Rocha, Kon, Milojicic, & Meirelles, 2019). Traditional teaching methods, such as lectures and textbook readings, may not be sufficient to help students develop the practical skills and problem-solving abilities required to succeed in a DevOps engineering role (Bruel & Jiménez, 2019; Ferino et al., 2021, 2023; Fernandes et al., 2020, 2022; Hobeck et al., 2021; Jones, 2019a). Accordingly, a gap between education and industry exists in certain DevOps practices (Sánchez-Cifo, Bermejo, & Navarro, 2023). Therefore, innovative teaching approaches that promote active learning and practical application are needed.

Implementing a system-based learning (SBL) approach for teaching DevOps engineering is justified by the need to foster a deeper understanding of the intricate relationships between various stages of the software development life cycle. In contrast to traditional methods, SBL encourages learners to view the entire system as a whole, promoting a seamless integration of development and operations processes (Harden, Davis, & Crosby, 1997; Spain, 2019). This holistic perspective not only helps students grasp the underlying principles of DevOps but also equips them with the critical thinking and problem-solving skills necessary to address complex, real-world challenges. Furthermore, collaboration and communication, which are central tenets of DevOps, are inherently promoted by the SBL approach (Halder, Joshi, Mehrotra, Rathinam, & Shrivastava, 2018). By fostering a learning environment that requires students to work together and engage in active dialogue, SBL cultivates a collaborative mindset that is indispensable for success in the field of DevOps engineering. Thus, an SBL approach is not only aligned with the core values and objectives of DevOps but also offers a more effective means of preparing students to excel in this burgeoning field.

Therefore, this article aims to explore the implementation of an SBL approach for teaching DevOps engineering as an elective course in an undergraduate software engineering program. We will discuss the theoretical foundations of SBL and its potential benefits for teaching DevOps engineering. We will also describe a case study in which we implemented SBL for a DevOps engineering course and evaluate its effectiveness based on student feedback and performance metrics. The findings of this study will provide insights into the use of SBL for teaching DevOps engineering and contribute to the development of innovative teaching approaches in software engineering education.

2 Background

2.1 DevOps Software Development Process

DevOps engineering has emerged as a critical approach to software development that emphasizes collaboration, automation, and continuous improvement of new software versions while guaranteeing their correctness and reliability (Bass, Weber, & Zhu, 2015; Leite et al., 2019; Zarour, Alhammad, Alenezi, & Alsarayrah, 2020). It involves the integration of development and operations teams to automate and streamline the software development process while ensuring high-quality software delivery (Hornbeek, 2019). As such, it has become an essential skill for software engineers seeking to succeed in today's fast-paced technology industry (Gurcan & Cagiltay, 2019). Nowadays, the field of DevOps engineering has rapidly evolved, transforming the landscape of software development and deployment by integrating the principles of CI, continuous delivery, and automation (Buttar et al., 2023). As a result, there is an increasing demand for skilled professionals who can effectively navigate and apply these principles in real-world contexts (Amaro, Pereira, & da Silva, 2022) and understand the culture and mindset of DevOps (Jha et al., 2023). In academia, unfortunately, academics are not motivated to learn or adopt DevOps, and no strong evidence exists of academics teaching DevOps (Pang, Hindle, & Barbosa, 2020). Few trials are reported in the literature that discuss the experience of offering a dedicated DevOps course at different universities (see, e.g., Demchenko et al., 2019; Hobeck et al., 2021; Jennings & Gannod, 2019; Paez & Fontela, 2023; Radenković, Popović, & Mitrović, 2022). Hence, to meet the increasing demand for DevOps professionals and to enrich the literature with more reports on offering DevOps courses, we have introduced DevOps engineering as an elective course in our software engineering program.

The development and operations teams create, deploy, and manage both the infrastructure (environments) and deployment (CI/CD) pipelines in a way that drives the DevOps culture, values, and practices (Díaz et al., 2024). A typical DevOps CI/CD pipeline consists of four main phases, as depicted in Figure 1. The source code repository is where the code for the application is stored and managed using a version control management system. It can be a public or private repository, such as GitHub or BitBucket. The CI stage takes the code from the repository and builds it into a binary artifact. This stage focuses on early and frequent integration of code changes. The resulted artifact is typically a Docker image or a Kubernetes pod. The continuous delivery/continuous deployment (CD) stage then tests the binary artifact and deploys it to a staging environment. The continuous delivery (CD) and continuous deployment (CD) are both DevOps practices aimed at streamlining the software development and release process. Continuous Delivery: Ensures software is always release-ready. Continuous Deployment: Automates the entire release process, including deployment to production. This allows the developers to test the changes in a production-like environment before deploying them to production. The continuous deployment stage then automatically deploys the binary artifact to the production environment. Although continuous delivery and continuous deployment are closely related concepts in DevOps, they differ in their final step of delivering changes to production.

The CI/CD pipeline can be implemented using a variety of tools and technologies (Alnamlah, Alshathry, Alkassim, & Jamail, 2021; Kamath, Vignesh, & Darshan, 2023). Some popular CI/CD tools include Jenkins, Travis CI, and CircleCI. Some popular CD tools include Docker and Kubernetes. The CI/CD pipeline is a critical part of the DevOps workflow. It helps to ensure that code is always up-to-date and that changes are deployed to production in a safe and controlled manner. The CI/CD pipeline can be customized to fit the specific needs of the application. For example, the pipeline can be configured to run different tests at different stages. The pipeline can also be configured to deploy the application to different environments, such as staging, production, and development. Figure 2 illustrates how the CI/CD pipeline intersects with the development and operation processes.

Hence, adopting a successful DevOps process in software development needs more effort from both academicians and professionals in the industry to better understand and implement DevOps practices and uncover neglected or untouched areas yet, for instance, controlling cross-functional teams in the dynamic and iterative nature of the control in DevOps process (Wiedemann, Wiesche, Gewald, & Krcmar, 2023).

2.2 SBL

SBL involves the integration of theory, practice, and reflection in a problem-based learning environment (Harden et al., 1997; Matinho et al., 2022; Syeed, Shihavuddin, Uddin,

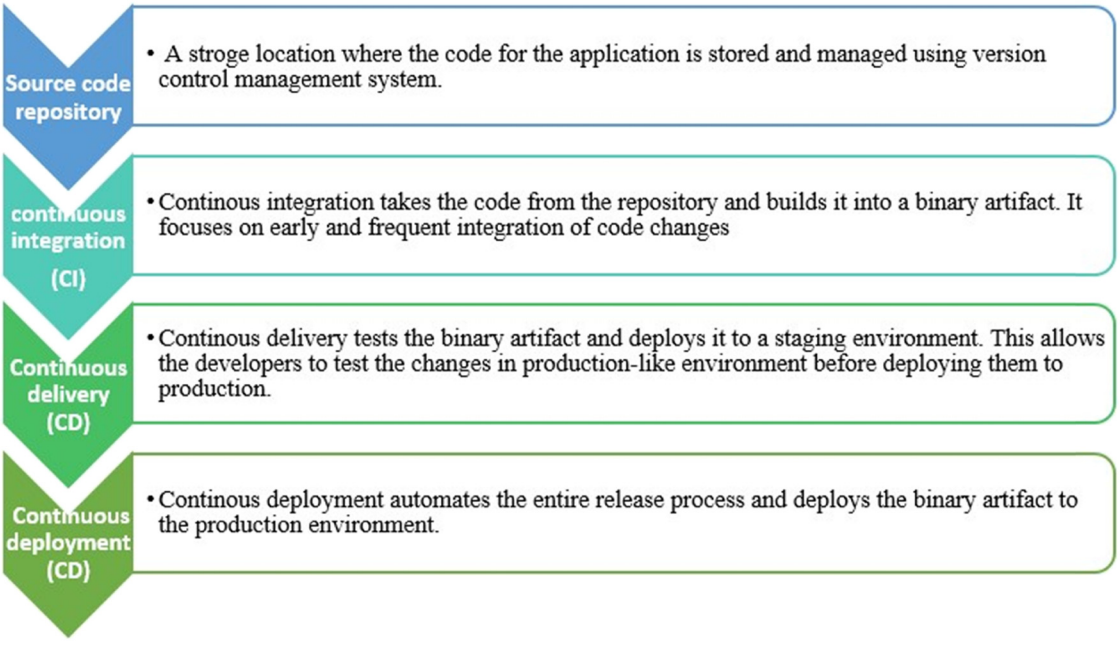

Figure 1: Typical DevOps CI/CD pipeline.

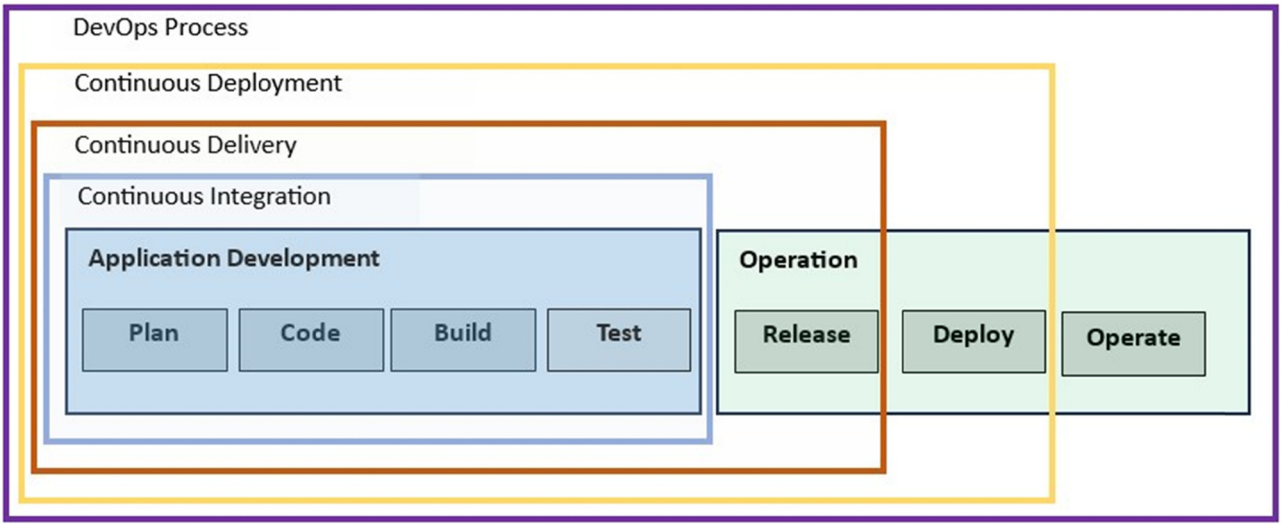

Figure 2: DevOps CI/CD pipeline intersection with development and operation processes.

Hasan, & Khan, 2022). SBL is not uncommon in graduate education; it is being used in medicine to understand the interconnectedness of bodily systems, which is crucial for doctors and nurses. For example: Zhong, Huang, and Lin (2023) found that the application of the OBL (organ system-based learning) practice teaching model in the digestive system department is conducive to improving the theoretical knowledge of nursing students and improving the core competence and clinical teaching satisfaction. Similarly, Lu, Li, Cao, and Min (2022) found that the OBL model in undergraduate clinical practice teaching of anesthesiology has significantly improved the education quality, theoretical achievement, and comprehensive ability of interns. SBL emphasizes the use of real-world scenarios and hands-on experiences to help students develop the knowledge and skills required to solve complex problems (Namasivayam, Fouladi, Tien, & Moganakrishnan, 2019). By simulating a real-world environment, students can practice problem-solving skills and apply theoretical concepts to practical scenarios. Garrubba, Donkers, Daniel, and Ennulat (2015) found that both system-based and problem-based are both effective teaching methods for teaching physical diagnosis curriculum. Atta and AlQahtani (2018) found that teaching pathology, by traditional medical schools that are in the process of shifting toward an integrated SBL, needs to be more documented and addressed. As a result, the system-based practice has become increasingly common in medical education (Bhate et al., 2023; Castillo et al., 2020).

Accordingly, SBL, which emphasizes the integration of knowledge across disciplines and the development of practical problem-solving skills, offers a more holistic and immersive learning experience well-suited to meet the demands of the fast-paced and interconnected world of DevOps engineering.

While SBL and project-based learning (PBL) share similarities, such as their student-centered and real-world problem-focused nature, they also exhibit notable distinctions (Li & Zhu, 2023; Radenković et al., 2022). SBL emphasizes the comprehension of complex systems (Purao, Vaishnavi, Welke, & Lenze, 2009), adopts an interdisciplinary approach, and places greater emphasis on collaborative learning (Raj et al., 2021), e.g., focuses on system-thinking perspective (Ahlgren, 2013; Finn, Avalos, & Dunne, 2014; Kim & Senge, 1994; Mobus & Kalton, 2015; Spain, 2019). On the other hand, PBL primarily focuses on resolving specific problems and may exhibit a narrower scope compared to SBL. SBL projects tend to be more extensive and intricate, allowing for diverse interpretations and solutions, while PBL projects often have more defined parameters. In the context of software engineering, SBL offers a promising approach to teaching intricate subjects such as DevOps engineering. DevOps engineering involves the integration of software development and operations teams to automate and streamline the software development process while ensuring the delivery of high-quality software. Proficiency in this field requires a range of skills, including knowledge of CI/CD pipelines, containerization, and IaC. Table 1 summarizes the benefits and challenges of various teaching strategies that are used in various domains including software engineering and can be adopted to teach DevOps courses. In this research work, we are investigating the adoption of the SBL strategy in developing software applications from the requirements engineering and analysis to the testing and then delivery. Further research is needed to evaluate the

effectiveness of other approaches and identify best practices for preparing students with the knowledge and skills necessary to succeed in the DevOps field.

3 Methodology

Implementing SBL in DevOps courses can be accomplished by focusing on the following key aspects:

- The system as a whole: SBL teaches students to perceive the software development process as a holistic system rather than a collection of independent tasks. This approach enhances their understanding of how different parts of the process interact and how modifications in one part can affect others.
- The environment: SBL instructs students to consider the environment in which software will be developed and deployed. This includes factors such as the hardware, software, and people involved in the process.
- The people: SBL guides students in developing effective teamwork skills, fostering their ability to work collaboratively with others. This involves cultivating communication, collaboration, and conflict-resolution skills, which are vital in a DevOps engineering role.

As the concept of SBL adoption in DevOps course delivery is relatively new, we will use case study

Table 1: Comparison of well-known teaching approaches

Approach	Benefits	Challenges	References
PBL	• Hands-on experience • Collaboration • Communication • Problem-solving • Active learning • Increased engagement • Deeper understanding	• Project design/management • Resource and support needs • Time commitment • Learning objective coverage	Adorjan and Solari (2021), Ceh-Varela, Canto-Bonilla, and Duni (2023), Mielikäinen, Viippola, and Tepsa (2023), Naik and Girase (2020), Pérez and Rubio (2020)
Simulations and case studies	• Safe learning environment • Reflection and analysis • Critical thinking • Real-world exposure • Resource-independent • Adaptable skill levels	• Real-world complexity capture • Simulation setup time • Case study selection/adaptation • Limited hands-on experience	Campos, Nogal, Caliz, and Juan (2020), de França and Travassos (2004), de França and Ali (2020), Lee and Rine (2004)
Game-based learning	• High engagement and motivation • Experimentation and risk-taking • Problem-solving and critical thinking • Personalized learning • Collaboration and teamwork	• Development effort/investment • Aligned game mechanics • Student suitability • Assessment difficulty	Baumann (2020), Elina Jääskä and Aaltonen (2023), Flores, Paiva, and Cruz (2020), Gomes and Lelli (2021), Ngandu, Risinamhodzi, Dzvapatsva, and Matobobo (2023), Videnovik, Vold, Kiønig, Bogdanova, and Trajkovik (2023)
Blended learning	• Flexibility and adaptability • Personalized learning • Diverse learning styles • Approach strengths combined • Active learning and engagement	• Planning and coordination • Integration and outcome assurance • Increased instructor workload • Technology intensiveness	Barbosa (2022), Luzik, Akmaldinova, and Tereminko (2019), Mielikäinen et al. (2023), Ożadowicz (2020), Perez, Castellanos, and Correal (2020)
SBL	• Deeper Understanding of the system as a whole • Enhanced students' Problem-Solving skills • Enhance student's BigPicture Thinking • Promotes effective Collaboration in interdisciplinary teams	• Teacher Role Shift from knowledge transmitters to facilitators • Curriculum Development and Design of learning experiences • Evaluation Strategies where new assessment methods that go beyond traditional memorization-based tests • Potential Abstraction of the complex systems	Ackoff (1973), Dul et al. (2012), Salmon et al. (2017), Sharma (2006), Walker et al. (2010, 2016), Wilson (2014), Woods and Dekker (2000)

methodology to explore how SBL can be adopted in a DevOps course. A case study is a research methodology that investigates a single unit, such as a person, group, organization, or event. It is an in-depth study that seeks to understand the phenomenon in its real-world context (Hunziker & Blankenagel, 2024; Moeed, Dobson, & Saha, 2024; Verma, Verma, & Abhishek, 2024). Case studies are often used in software engineering research to investigate complex problems or to understand the impact of new technologies. Among the various types of case studies, we are adopting an exploratory case study approach, which is not uncommon in software engineering; for instance, Dingsøyr, Nerur, Balijepally, and Moe (2012) employed an exploratory case study approach to provide insights into the evolution and understanding of agile methodologies in the software development industry. Helo and Hao (2022) have conducted an exploratory case study to analyze the emerging AI-based business models of different case companies aiming to identify several areas of value creation for the application of AI in the supply chain. Hata, Novielli, Baltes, Kula, and Treude (2022) to understand how developers use the new GitHub asking questions feature, how the developers perceive it, and how it impacts the development processes. In our context, using the case study methodology involves three main phases (Figure 3).

3.1 Preparation

- Define the research questions: The first step is to define the research questions. These research questions are aligned with the principles of SBL. The main research questions to tackle in this case study are as follows:
  1. RQ1: How can an SBL approach be implemented in a DevOps course?
  2. RQ2: How does this approach impact student learning outcomes compared to the traditional delivery method?
  3. RQ3: What are the perceived benefits and challenges of using SBL for teaching DevOps concepts?
- Identify the learning objectives and key concepts: The main objective of this case study is to teach students the DevOps process in a way that they perceive the software development process as a whole system, rather than as a collection of independent tasks. The key concepts that students need to learn include communication and collaboration, CI, continuous delivery, and the importance of automation.
- Design the course: Design the course includes

1. Select a specific course to deliver the DevOps material. In our case, we have selected an elective course in the software engineering curricula for senior students (400-level courses) to offer the DevOps course.
2. Design learning activities: The next step is to design learning activities that will help students to learn the key concepts. These activities should be hands-on and should encourage students to work together. The main activity focuses on involving students in projects to develop a software system using DevOps principles.

3.2 Implementation

- Assess the current state of knowledge: It is important to assess the current state of knowledge related to the main DevOps concepts before starting the course. This will help you to identify any gaps in their knowledge and to tailor the course to their specific needs. In our case, and during the first class, the instructor asked the students to discuss their understanding of the DevOps process, the CI/CD concept, and the main automation tools. The students knowledge of these concepts was almost null. Few students mentioned that they had heard of the term DevOps but had no clue about how it works.
- Deliver the course: At this phase, the DevOps course, which has been prepared, see next section, by the authors of this article, is delivered to students.
- Provide feedback: It is important to provide feedback to students throughout the course. This will help them to track their progress and to identify any areas where they need additional help.
- Update course as needed: As you teach the course, you may need to modify the learning activities or the assessment methods to ensure that the course is meeting the needs of the students.

3.3 Evaluation

- Assess student learning: Finally, assess student learning to ensure that they have achieved the learning objectives. This assessment can be done through a variety of methods, such as quizzes, exams, and projects.
- Reflect on the course: It is important to reflect on the course after it is finished. This will help you to identify what worked well and what could be improved for future courses.

4 Results

This section discusses the implementation of the presented method in the previous section to design and run the DevOps course using the SBL approach.

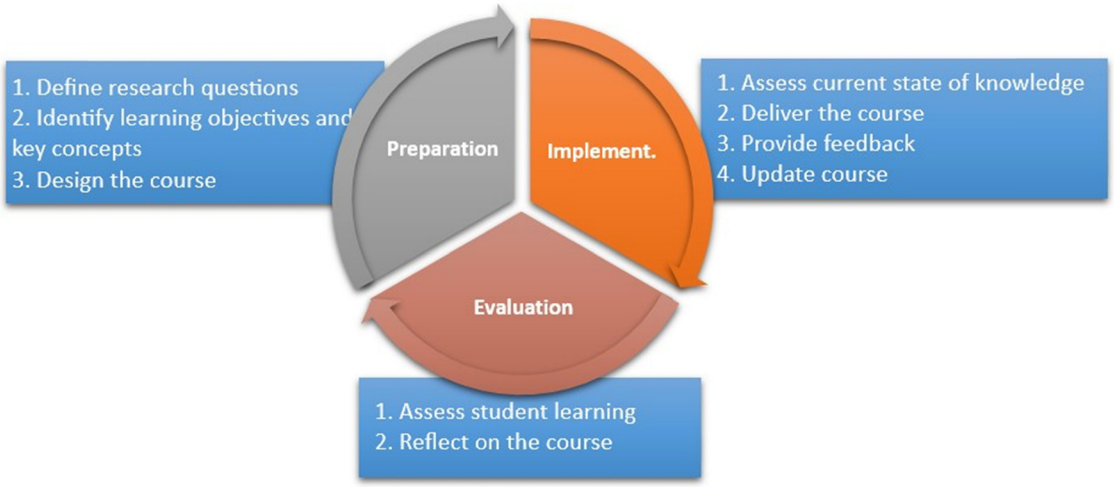

Figure 3: Research methodology steps.

4.1 Step 1: Preparation

The course "DevOps Engineering" is meticulously designed to equip students with the requisite knowledge and skills to implement DevOps as a method for delivering systems to enterprises with enhanced quality and velocity. To implement an SBL approach in a DevOps course, the following steps are taken:

1. Define what the System is? Developing a software system using the DevOps development approach is an ideal candidate for SBL because it inherently functions as a system. The system in the DevOps process consists of:
   (a) Development (coding, testing, building, and version control).
   (b) Operations (infrastructure and configuration management).
   (c) Automation (test automation, CI/CD, and monitoring automation).
2. Design Curriculum and Learning Activities: The course is structured into ten modules, each focusing on a unique aspect of DevOps engineering. Each module includes succinct videos (2–5 min) that elaborate on each concept in detail. Moreover, a minimum of two detailed real-world case studies per module are incorporated, offering practical perspectives related to the module's concepts. In the tenth week of the semester, two guest lecturers from the industry are invited to give a lecture (one for each of them) for students and answer their questions about the practical use of DevOps in real environments. The course adopts PBL where students work in teams (4–5 students each) to deliver their final project where they work together to build and deploy a software application using DevOps principles. Students in each project select an active open-source project and build a pipeline for it. The project was divided into three phases to make it more manageable and achievable for students. The course's grade distribution is based on assignments and labs (20%), presentations (5%), a midterm exam (15%), a course project (20%), and a final exam (40%). During the course delivery, three assignments and nine labs are practiced by students. The labs practically guide students through the construction of a complete pipeline, from version control systems to exploring KubeCTL. They cover an array of topics such as version control systems, CI, containerization, configuration management, and continuous monitoring. The course targets six specific learning outcomes (CLOs), namely:
   (a) CLO 1: Recognize DevOps principles, methods, and practices.
   (b) CLO 2: Appraise DevOps principles and practices including integration, delivery, and testing.
   (c) CLO 3: Illustrate how to apply DevOps tools used for monitoring, alerting, and reporting.
   (d) CLO 4: Apply DevOps concepts in an enterprise environment by automating processes using tools.
   (e) CLO 5: Select the best strategy to deploy software applications utilizing CI/CD pipelines.
   (f) CLO 6: Apply the principles of DevOps to a software development project.
3. Promote collaboration and run feedback loops. The project and assignments' activities encourage collaboration and communication among students. This reflects the real-world scenario where students play roles like developers, operations team members, and automation specialists and work together effectively. Students also practice the feedback loops where, for example, they analyze how

a code change in development triggers the CI/CD pipeline to respond to that code change.

By achieving the specified outcomes and getting involved in the various activities and project work, students will be well-equipped with the necessary knowledge and skills to proficiently implement DevOps practices and principles in a real-world software development environment. Defining course objectives, course learning outcomes, course material, assignments, assessments, and project design that involves adopting the DevOps process to develop and release a whole system based on the SBL approach, all these activities provide an answer for the first research question RQ1: How can an SBL approach be implemented in a DevOps course?

4.2 Step 2: Implementation

The DevOps engineering course has been delivered twice in two consecutive semesters: Fall 2022 and Spring 2023. The section size ranges from 20 to 25 students in each semester. As presented in the preparation step, the course material and labs are delivered in 15 weeks per semester. As planned, the project runs through three main phases as follows:

1. In the first phase, students familiarize themselves with the chosen project's specifics, including understanding its architecture and design. They provide the GitHub link to the project for transparency and accessibility.
2. The second phase shifts focus to the practical application of DevOps principles. Students undertake tasks such as local testing, code pushing into production, setting up a CI/CD pipeline, monitoring, and application deployment, thereby providing a hands-on introduction to the critical components of DevOps engineering.
3. The third phase requires students to execute another cycle of their pipeline, incorporating a new deployment with a new feature or a bug fix. This allows them to gain experience in iterative development and CI, essential facets of DevOps. They also compile a final project report and presentation, enhancing their communication and documentation skills.

The project was delivered on a dedicated GitHub public repository, which contained the project's source code, DevOps setup related to the project, and project report (ideally as a set of linked markdown documents starting from the repository README). The project report described the project rationale and contained all artifacts related to the problem analysis and the design phases.

At the project's culmination, students defend their design and implementation in an oral examination. Each student is allocated a 5-min presentation slot, with a maximum total presentation time of 20 min per group. The examination is an opportunity for students to articulate their understanding of the project, defend their design decisions, and demonstrate their practical skills. The evaluation, as discussed in step 3 below, considers the quality of the presentation, the project report, and the developed code.

To provide a comprehensive overview of the student's experience and the quality of their work, we will delve into two exemplary projects completed by the students. The first project, known as IEEE CMS, entails a content management solution designed to facilitate the management of various types of contents by university clubs. This particular version focuses on catering to the needs of event planners and organizers. The project aims to streamline the process of managing and updating content on the chapter's website and utilizes a CI/CD pipeline to automate the process of building, testing, and deploying the application. The project was undertaken by a team of five engineers, all serving as DevOps engineers who have worked together to build the CI/CD pipeline. Figure 4 showcases the CI/CD pipeline implemented for their project, and Figure 5 illustrates the deployment diagram associated with their project. The system consists of a container for a web server, another container for the application server, and a third container for the database server. The system also includes a CI/CD pipeline, which automates the process of building, testing, and deploying the application.

The second project centers on the development of a Python AI assistant. The Python AI assistant aims to create a chatbot that can assist with a variety of tasks such as scheduling meetings, answering questions, and providing information. The project entailed the creation of "Jarvis," a voice-command assistant service designed for Python 3.8. The team responsible for this project consisted of four engineers, all serving as DevOps engineers who have worked together to build the CI/CD pipeline. The project

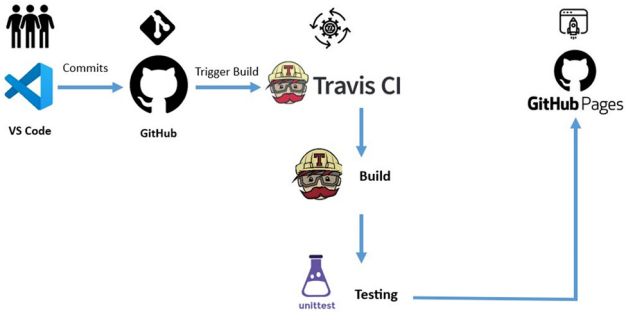

Figure 4: IEEE CMS CI/CD pipeline.

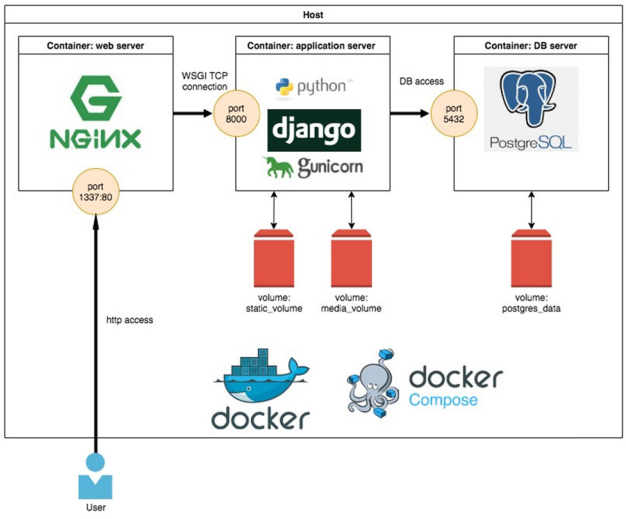

Figure 5: IEEE CMS deployment diagram.

utilizes a CI/CD pipeline to automate the process of building, testing, and deploying the chatbot. Figure 6 portrays the CI/CD pipeline implemented for their project. Furthermore, Figure 7 provides a dashboard that showcases the status of each build. It provides a visual representation of the build status of the application, allowing the development team to quickly identify any issues or errors that may have occurred during the build process. The dashboard displays various metrics such as the build status, test coverage, and deployment status.

4.3 Step 3: Evaluation

The course exit survey is an important tool to gather valuable feedback from students about their academic experience. It is designed to evaluate the level of student satisfaction with the curriculum and course being taught, as well as their confidence in achieving the course learning outcomes. Students were asked to use a five-category scale, ranging from "strongly disagree" (1) to "strongly agree" (5), to evaluate statements on their achievement level of each CLO defined for this course. Their feedback is an indicator of whether the adopted SBL teaching approach was effective or not. The students, who filled out the survey, were of the same level (senior students), and all of them had no experience with SBL teaching approach and that they had not practiced it before. Therefore, we can assume that the SBL was fairly new educational approach for all the students in this study. The average age for the students was 22 years old and all of them were male.

The course exit survey was conducted twice. The total number of students' feedback for the two semesters was 45 responses (n = 45), and they were studied as one sample to analyze their responses and see how effective was the SBL teaching strategy, see Figure 8 for details about the overall achievement per CLO in each semester.

The summative descriptive statistics are shown in Table 2; more than 70% of the students indicated that they were "Adequately Satisfied" or "Fully Satisfied" with

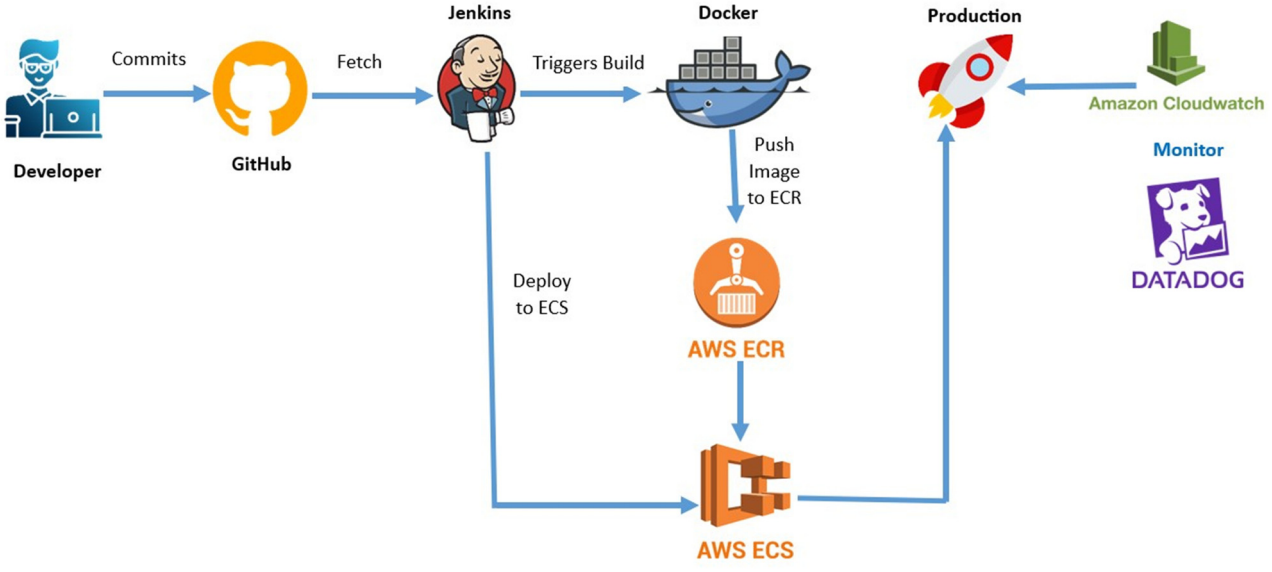

Figure 6: Python AI CI/CD pipeline.

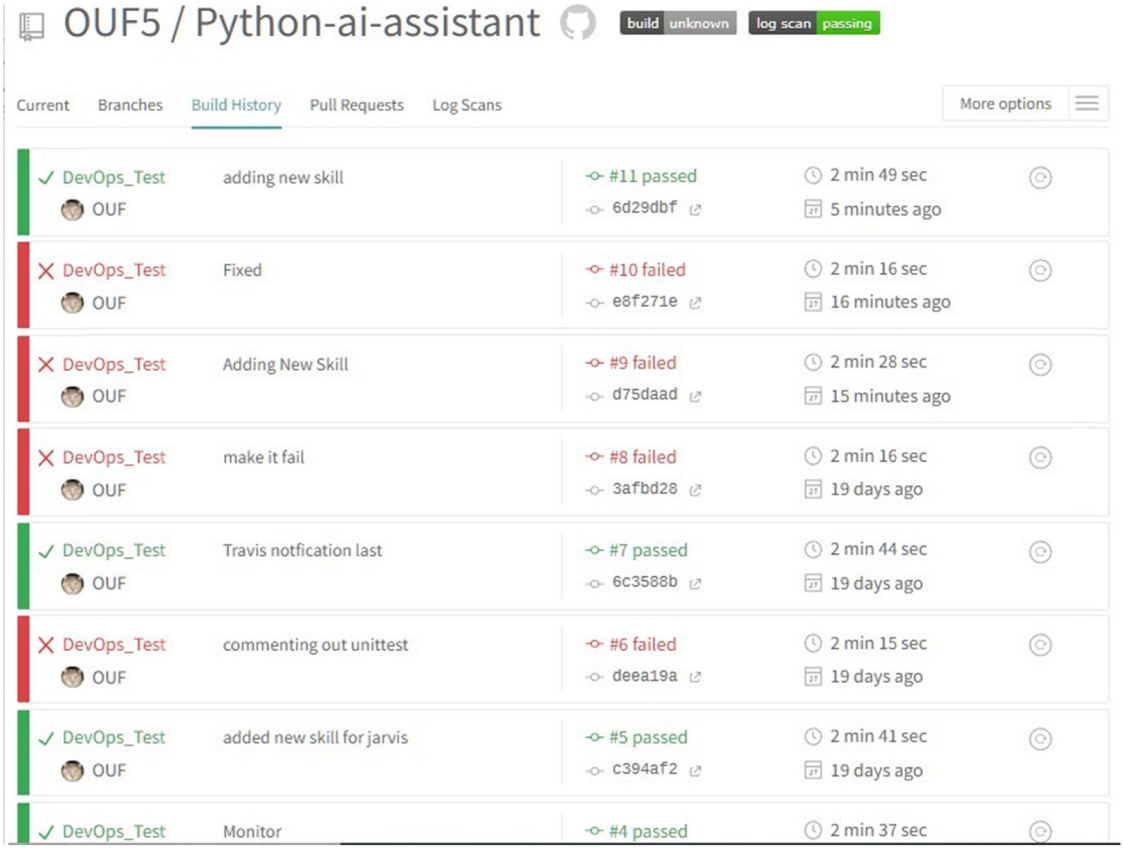

Figure 7: Python AI build status dashboard.

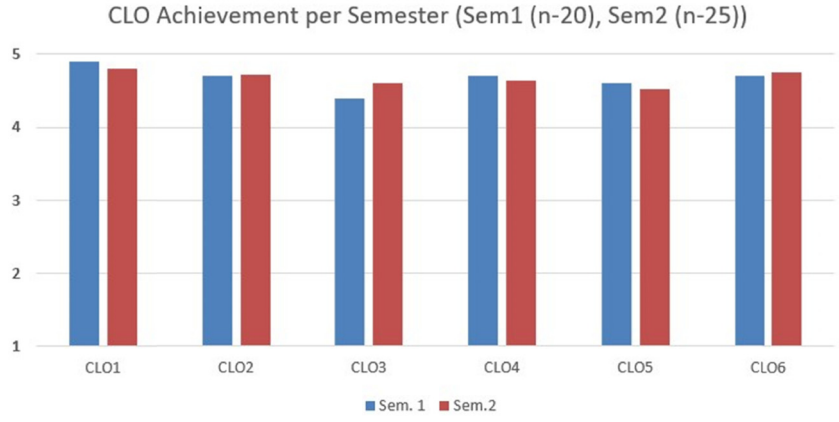

Figure 8: Course learning outcomes – students' achievements.

Table 2: Student responses to the CLO exit survey (n = 45)

CLO	Not satisfied (%)	Barely satisfied (%)	Somewhat satisfied (%)	Adequately satisfied (%)	Fully satisfied (%)	Mean	STD Dev.
CLO 1	0	0	2	11	87	4.84	0.34
CLO 2	0	0	9	11	80	4.71	0.30
CLO 3	2	4	9	9	76	4.51	0.28
CLO 4	0	4	9	13	76	4.67	0.28
CLO 5	0	4	9	13	73	4.56	0.27
CLO 6	0	0	4	18	78	4.73	0.30

their accomplishment of each course learning objective. This suggests that the learning process was adequately successful. CLO 1 and CLO 2 have the highest percentage of students (87 and 80%, respectively) reporting "Fully Satisfied." This suggests that the course effectively equipped students with core DevOps skills. We noticed that there's a slight decrease in the percentage of "Fully Satisfied" responses as we move from CLO 1 and 2 to CLO 3–6 (ranging from 73 to 78%). This might indicate a need for further refinement in how these later learning outcomes are addressed in the course. The low percentages (0–4%) in the "Not Satisfied" and "Barely Satisfied" categories suggest that most students felt the course adequately addressed the learning objectives.

Note that the standard deviation values for all CLOs are relatively low (between 0.27 and 0.34). This suggests a certain level of consistency in student responses, meaning the results are not overly skewed towards one satisfaction level or another. Overall, these results suggest a successful DevOps course with a strong foundation in core skills (CLO 1 and 2) and a generally positive student experience. However, there might be room for improvement in how the course addresses learning outcomes (CLO 3–6) to maintain the same level of high student satisfaction. The number of students who passed the course compared to the number who registered for the course is shown in Figure 9. The passing rate per semester is shown in Figure 10, and the detailed grade distribution of all students is also shown in Figure 11. It is worth mentioning that while preparing this article, a well-known local organization interviewed our graduates who have taken the DevOps course and were totally satisfied with their DevOps knowledge and skills and the job interviews resulted in hiring all those graduates. The discussions above answer the second research question: How does this approach impact student learning outcomes compared to a traditional delivery method?

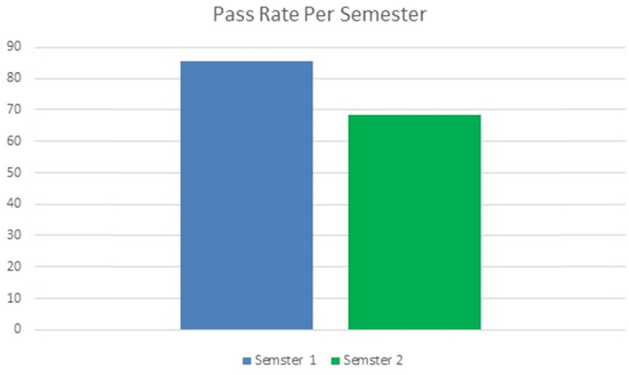

Figure 10: Passing rate per semester.

4.4 Educational Recommendations

The results of the conducted study suggest that SBL can be applied to facilitate DevOps teaching courses. Based on our findings, the learning of important but rather complex topics, such as DevOps, can be fostered when students are able to apply and practice learning objectives by implementing the whole software development process as one system.

The key lessons learned and presented in the following section will help to understand and guide the design of an SBL solution to educate project management phenomena and practices. We assume that these lessons are not specific to teaching project management methods, like DevOps, but can be applied to the design and implementation of other learning solutions outside of the scope of project management as well. Key lessons learned are:

1. Ensure that the DevOps theoretical bases and SBL learning approach are explained clearly through lectures, reading materials, videos, and case studies.
2. Analyze the course content and delivery methods for CLO 3–6 to identify areas for improvement.

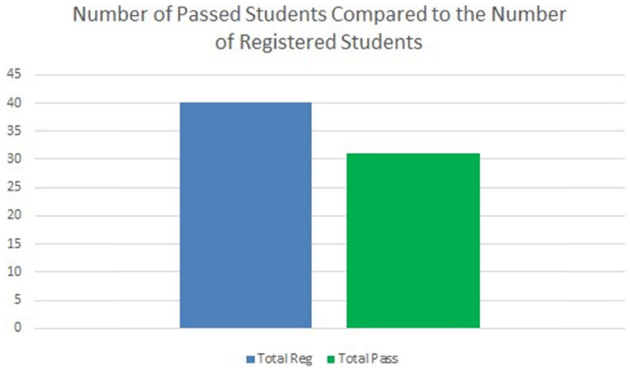

Figure 9: Passing rate of registered students.

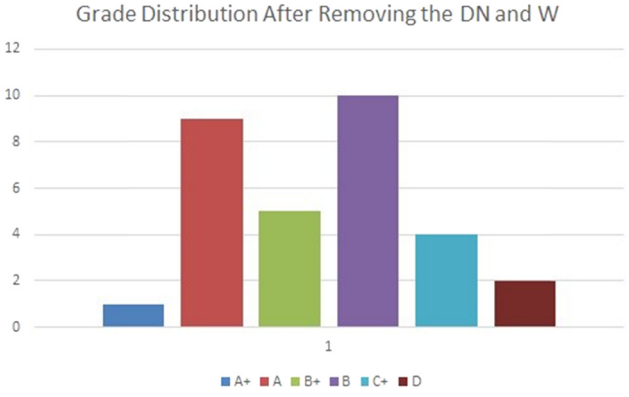

Figure 11: Course learning outcomes – students' achievements.

3. Explain DevOps pipeline phases and development requirements, and rules before start using the tools and practice building the pipeline so that the learning focus will be on the technical aspects, not the theoretical aspects.
4. Build repetitions into various DevOps projects and solutions: Repetition helps students memorize and learn.
5. Conduct follow-up interviews with students (especially those who reported lower satisfaction) to gain more specific feedback about their learning experience and refine the DevOps course to ensure that all learning outcomes are effectively addressed.
6. We suggest not waiting until the end of the course to gather feedback. In the future iterations of DevOps course delivery, we plan to conduct mid-course surveys or interviews to identify areas where students might need additional support.

5 Challenges, Limitations, and Future Work

While SBL is found to be a powerful approach for teaching DevOps course, we faced some challenges when implementing it:

1. Instructor's role: Traditional instructors are often knowledge transmitters. SBL requires a transition to facilitator roles, guiding students through system exploration and analysis. This might require faculty development to equip instructors with the necessary skills for facilitating SBL activities effectively.
2. Course design: Designing DevOps learning experiences that represent complex software engineering systems and their interactions necessitates careful planning. Developing new course materials, labs, and case studies specifically tailored to SBL principles in the domain of DevOps was time-consuming and resource-intensive.
3. Students achievement evaluation: Evaluating student understanding of DevOps principles and automation requires new assessment methods that are beyond traditional memorization-based tests. In our case, we adopted project-based assessments that can demonstrate a student's ability to apply SBL principles to design, develop, and deploy software within a system context. This approach was found to be complex and time-consuming yet effective in assessing students' achievement.
4. Time limitations: SBL in the domain of DevOps requires specific software tools and cloud infrastructure. Such resources may not always be available, hence, instructors need to invest more time to explore open-source alternatives and consider creative solutions to maximize available resources.

The present study has some limitations, which may limit the generalization of its findings. Our SBL learning approach focuses on teaching DevOps practices to senior undergraduate students. Our findings are based on some basic measures calculated based on students' survey results. For future investigation purposes, more case studies need to be conducted on using SBL in teaching various software engineering courses. Hence, more measures and variance analysis could be incorporated into the learning solutions.

In conclusion, the SBL approach for teaching DevOps engineering presented in this article not only addresses the limitations of traditional teaching methods but also bridges the gap between academic learning and practical application. The introduction of the "DevOps Engineering" course as an elective in a BS software engineering program has proven to be a significant step towards the practical integration of systems thinking and engineering principles into the software development process. The positive response from students and the observable improvement in their problem-solving skills and comprehensive understanding of the system suggest the effectiveness of this approach. This provides an answer to the third research question: What are the perceived benefits and challenges of using SBL for teaching DevOps concepts?

Future work should focus on refining the course curriculum based on student feedback and industry trends, DevOps is continually evolving to ensure its relevancy and effectiveness. Additionally, considering the dynamic nature of DevOps engineering, the curriculum should be updated regularly to incorporate emerging tools and practices. Finally, extending this study to other universities and comparing the results can further validate this teaching approach, contributing significantly to the pedagogy of DevOps engineering. Future iterations of the course could consider incorporating more advanced topics in DevOps, such as machine learning operations or data operations, to reflect the expanding role of DevOps in these areas. Additionally, to further enhance the learning experience, more interactive and collaborative learning methods should be employed. Moreover, the course could benefit from partnerships with industry professionals. Guest lectures or workshops led by industry experts could provide valuable insights into the practical application of DevOps principles and practices and offer students the opportunity to network with professionals in the field. Finally, it is crucial to maintain a robust feedback mechanism for continuous improvement of the course. Regular student feedback should be solicited and used to inform future course improvements and updates. This will

ensure that the course remains relevant, engaging, and effective in achieving its learning outcomes.

Acknowledgment: The authors would like to acknowledge the support of Prince Sultan University for paying the Article Processing Charges of this publication.

Author contributions: All authors made significant contributions to the research and manuscript preparation. All authors have given their approval for the final version of the manuscript and agree to be accountable for the work.

Conflict of interest: The authors state no conflict of interest.

References

Ackoff, R. L. (1973). Science in the systems age: Beyond ie, or, and ms. *Operations Research*, *21*(3), 661–671.

Adorjan, A., & Solari, M. (2021). Software engineering project-based learning in an up-to-date technological context. *In 2021 IEEE URUCON*, pp. 486–491. doi: 10.1109/URUCON53396.2021.9647348.

Ahlgren, E. (2013). How to teach systems in engineering education: The case of an energy systems course. *Proceedings of the IETEC'13 Conference, Ho Chi Minh City, Vietnam*.

Alnamlah, B., Alshathry, S., Alkassim, N., & Jamail, N. (2021). The necessity of a lead person to monitor development stages of the DevOps pipeline. *Indonesian Journal of Electrical Engineering and Computer Science*, *21*(1), 348.

Amaro, R., Pereira, R., & da Silva, M. M. (2022). Capabilities and practices in DevOps: A multivocal literature review. *IEEE Transactions on Software Engineering*, *49*(2), 883–901.

Atta, I. S., & AlQahtani, F. N. (2018). Mapping of pathology curriculum as quadriphasic model in an integrated medical school: How to put into practice? *Advances in Medical Education and Practice*, 549–557.

Barbosa, M. W. (2022). Using blended project-based learning to teach project management to software engineering students. *International Journal of Mobile and Blended Learning*, *14*(1), 1–17

Bass, L., Weber, I., & Zhu, L. (2015). *Devops: A software architect's perspective* (1st). AddisonWesley Professional.

Baumann, A. (2020). Teaching software engineering methods with agile games. *2020 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1550–1553). doi: 10.1109/EDUCON45650.2020.9125129.

Bhate, T. D., Sukhera, J., Litwin, S., Chan, T. M., Wong, B. M., & Smeraglio, A. (2023). Systemsbased practice in graduate medical education: Evolving toward an ideal future state. *Academic Medicine*, *99*(4), 357–362.

Bruel, J. M., & Jiménez, M. (2019). Devops' 18 education panel: Teaching feedback and challenges. *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5–6, 2018, Revised Selected Papers 1* (pp. 221–226).

Buttar, A. M., Khalid, A., Alenezi, M., Akbar, M. A., Rafi, S., Gumaei, A. H., & Riaz, M. T. (2023). Optimization of DevOps transformation for cloud-based applications. *Electronics*, *12*(2), 357.

Campos, N., Nogal, M., Caliz, C., & Juan, A. A. (2020). Simulation-based education involving online and on-campus models in different european universities. *International Journal of Educational Technology in Higher Education*, *17*, 107–189. doi: 10.1186/s41239-020-0181-y.

Castillo, E. G., Isom, J., DeBonis, K. L., Jordan, A., Braslow, J. T., & Rohrbaugh, R. (2020). Reconsidering systems-based practice: Advancing structural competency, health equity, and social responsibility in graduate medical education. *Academic Medicine*, *95*(12), 1817–1822.

Ceh-Varela, E., Canto-Bonilla, C., & Duni, D. (2023). Application of project-based learning to a software engineering course in a hybrid class environment. *Information and Software Technology*, *158*, 107–189. doi: 10.1016/j.infsof.2023.107189

Díaz, J., Pérez, J., Alves, I., Kon, F., Leite, L., Meirelles, P., & Rocha, C. (2024). Harmonizing DevOps taxonomies – a grounded theory study. *Journal of Systems and Software*, *208*, 111908.

de França, B. B., & Travassos, G. H. (2004). Simulation based studies in software engineering: A matter of validity. *CLEI Electronic Journal*, *18*(1), 4

de França, B. B. N., & Ali, N. B. (2020). The role of simulation-based studies in software engineering research. In *Contemporary empirical methods in software engineering* (pp. 263–287). Cham: Springer International Publishing. doi: 10.1007/978-3-030-32489-610.

Demchenko, Y., Zhao, Z., Surbiryala, J., Koulouzis, S., Shi, Z., Liao, X., & Gordiyenko, J. (2019). Teaching DevOps and cloud based software engineering in university curricula. *2019 15th International Conference on eScience (eScience)*, 548–552. doi: 10.1109/eScience.2019.00075.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of systems and software*, 85(6), 1213–1221.

Dul, J., Bruder, R., Buckle, P., Carayon, P., Falzon, P., Marras, W. S., ... Van der Doelen, B. (2012). A strategy for human factors/ergonomics: Developing the discipline and profession. *Ergonomics*, *55*(4), 377–395.

Elina Jääskä, J. K., & Aaltonen, K. (2023). A game-based learning method to teach project management – the case of the earned value management. *Cogent Education*, *10*(2), 1–22. doi: 10.1080/2331186X.2023.2264035.

Ferino, S., Fernandes, M., Cirilo, E., Agnez, L., Batista, B., Kulesza, U., ... Treude, C. (2023). *Overcoming challenges in DevOps education through teaching methods*. arXiv preprint arXiv:2302.05564.

Ferino, S., Fernandes, M., Fernandes, A., Kulesza, U., Aranha, E., & Treude, C. (2021). Analyzing DevOps teaching strategies: An initial study. *Proceedings of the XXXV Brazilian Symposium on Software Engineering* (pp. 180–185).

Fernandes, M., Ferino, S., Fernandes, A., Kulesza, U., Aranha, E., & Treude, C. (2022). DevOps education: An interview study of challenges and recommendations. *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training* (pp. 90–101).

Fernandes, M., Ferino, S., Kulesza, U., & Aranha, E. (2020). Challenges and recommendations in DevOps education: A systematic literature review. *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (pp. 648–657).

Finn, Y., Avalos, G., & Dunne, F. (2014). Positive changes in the medical educational environment following introduction of a new systems-based curriculum: Dreem or reality? curricular change and the environment. *Irish Journal of Medical Science*, *183*, 253–258.

Flores, N., Paiva, A. C. R., & Cruz, N. (2020). Teaching software engineering topics through pedagogical game design patterns: An empirical study. *Information*, *11*(3), 153. doi: 10.3390/info11030153.

Garrubba, C., Donkers, K., Daniel, L., & Ennulat, C. (2015). Perceptions of physician assistant students' readiness with system-based vs problem-based physical diagnosis curriculum. *The Internet Journal of Allied Health Sciences and Practice*, *13*(3), 9.

Gomes, R. F., & Lelli, V. (2021). Gamut: Game-based learning approach for teaching unit testing. *Proceedings of the XX Brazilian Symposium on Software Quality*. doi: 10.1145/3493244.3493263.

Gurcan, F., & Cagiltay, N. E. (2019). Big data software engineering: Analysis of knowledge domains and skill sets using lda-based topic modeling. *IEEE Access*, *7*, 82541–82552.

Halder, A., Joshi, A., Mehrotra, R., Rathinam, B., & Shrivastava, S. (2018). Setting objectives for a competency-based undergraduate obstetrics and gynecology curriculum. *Journal of Advances in Medical Education & Professionalism*, *6*(4), 147.

Harden, R., Davis, M., & Crosby, J. (1997). The new dundee medical curriculum: A whole that is greater than the sum of the parts. *Medical Education*, *31*(4), 264–271.

Hata, H., Novielli, N., Baltes, S., Kula, R. G., & Treude, C. (2022). Github discussions: An exploratory study of early adoption. *Empirical Software Engineering*, *27*, 1–32.

Helo, P., & Hao, Y. (2022). Artificial intelligence in operations management and supply chain management: An exploratory case study. *Production Planning & Control*, *33*(16), 1573–1590.

Hobeck, R., Weber, I., Bass, L., & Yasar, H. (2021). Teaching devops: A tale of two universities. *Proceedings of the 2021 ACM SIGPLAN International Symposium on SPLASH-E* (pp. 26–31). doi: 10.1145/3484272.3484962.

Hornbeek, M. (2019). *Engineering devops: From chaos to continuous improvement. and beyond*. BookBaby. https://books.google.com.sa/books?id=LRJAywEACAAJ.

Hunziker, S., & Blankenagel, M. (2024). Single case research design. In Research design in business and management: A practical guide for students and researchers (pp. 141–170). Wiesbaden: Springer Fachmedien Wiesbaden.

Jennings, R., & Gannod, G. (2019). DevOps – preparing students for professional practice. *2019 IEEE Frontiers in Education Conference (FIE)* (pp. 1–5).

Jha, A. V., Teri, R., Verma, S., Tarafder, S., Bhowmik, W., Kumar Mishra, S., … Philibert, N. (2023). From theory to practice: Understanding DevOps culture and mindset. *Cogent Engineering*, *10*(1), 2251758.

Jones, C. (2019a). A proposal for integrating DevOps into software engineering curricula. In J. M. Bruel, M. Mazzara, & B. Meyer (Eds.), *Software engineering aspects of continuous development and new paradigms of software production and deployment* (pp. 33–47). Cham: Springer International Publishing.

Jones, C. (2019b). A proposal for integrating DevOps into software engineering curricula. *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5–6, 2018, Revised Selected Papers 1*, 33–47.

Kamath, S., Vignesh, S., & Darshan, G. (2023). Revolutionizing cloud infrastructure management: Streamlined provisioning and monitoring with automated tools and user-friendly frontend interface. *2023 3rd International Conference on Intelligent Technologies (CONIT)* (pp. 1–6).

Kim, D. H., & Senge, P. M. (1994). Putting systems thinking into practice. *System Dynamics Review*, *10*(2–3), 277–290.

Lee, S. W., & Rine, D. C. (2004). Case study methodology designed research in software engineering methodology validation. In *Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering*, pp. 117–122.

Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A survey of DevOps concepts and challenges. *ACM Computing Surveys (CSUR)*, *52*(6), 1–35.

Li, X., & Zhu, W. (2023). The influence factors of students' transferable skills development in blended-project-based learning environment: A new 3p model. *Education and Information Technologies*, *28*(12), 16561–16591.

Lu, F., Li, P., Cao, J., & Min, S. (2022). Application of organ system based learning model in undergraduate clinical practice teaching of anesthesiology. *Chinese Journal of Medical Education Research*, 701–704.

Luzik, E., Akmaldinova, O., & Tereminko, L. (2019). Developing software engineering students' readiness forprofessional mobility through blended learning. *Advanced Education*, *6*, 103–111.

Matinho, D., Pietrandrea, M., Echeverria, C., Helderman, R., Masters, M., Regan, D., … McHugh, D. (2022). A systematic review of integrated learning definitions, frameworks, and practices in recent health professions education literature. *Education Sciences*, *12*(3), 165.

Meadows, D. H. (2008). *Thinking in systems*. Vermont: Chelsea green publishing.

Mielikäinen, M., Viippola, E., & Tepsa, T. (2023). Experiences of a project-based blended learning approach in a community of inquiry from information and communication technology engineering students at lapland university of applied sciences in finland. *E-Learning and Digital Media*, 20427530231164053.

Mobus, G. E., & Kalton, M. C. (2015). *Principles of systems science* (Vol. 519). Springer.

Moeed, A., Dobson, S., & Saha, S. (2024). *Research design and methodology. In Playful science investigations in early childhood: A longitudinal case study* (pp. 23–33). Singapore: Springer Nature Singapore.

Naik, V., & Girase, S. (2020). Project based learning methodology: An effective way of learning software engineering through database design and web technology project. *Journal of Engineering Education Transformations*, *34*, 375–379.

Namasivayam, S., Fouladi, M. H., Tien, D. T. K., & Moganakrishnan, J. A. S. (2019). Design Engineering as a Means to Enhance Student Learning in Addressing Complex Engineering Challenges. *Design Education Today: Technical Contexts, Programs and Best Practices* (pp. 249–270). Springer.

Ndaruhutse, S., Jones, C., & Riggall, A. (2019). *Why systems thinking is important for the education sector*. Berkshire: Education Development Trust.

Ngandu, M. R., Risinamhodzi, D., Dzvapatsva, G. P., & Matobobo, C. (2023). Capturing student interest in software engineering through gamification: A systematic literature review. *Discover Education*, *2*(47), 1–22. doi: 10.1007/s44217-023-00069-4.

Ożadowicz, A. (2020). Modified blended learning in engineering higher education during the covid19 lockdown – building automation courses case study. *Education Sciences*, *10*(10), 292. doi: 10.3390/educsci10100292.

Paez, N., & Fontela, C. (2023). Software engineering education in the DevOps era: Experiences and recommendations. *Anais do XXVI Congresso Ibero-Americano em Engenharia de Software* (pp. 130–137). doi: 10.5753/cibse.2023.24698.

Pang, C., Hindle, A., & Barbosa, D. (2020). Understanding DevOps education with grounded theory. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training* (pp. 107–118).

Pérez, B., & Rubio, A. L. (2020). A project-based learning approach for enhancing learning skills' and motivation in software engineering. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 309–315). doi: 10.1145/3328778.3366891.

Perez, B., Castellanos, C., & Correal, D. (2020). Measuring the quality of the blended learning approach to teaching computational sciences. *Journal of Physics: Conference Series*. doi: 10.1088/1742-6596/1587/1/012021.

Purao, S., Vaishnavi, V., Welke, R., & Lenze, L. (2009). A framework for problem-based learning of systems development and integration. *15th Americas Conference on Information Systems (AMCIS)*.

Radenković, M., Popović, S., & Mitrović, S. (2022). Project based learning for devops: School of computing experiences. *E-Business Technologies Conference Proceedings*, *2*, 127–131.

Raj, R., Sabin, M., Impagliazzo, J., Bowers, D., Daniels, M., Hermans, F., ... McCauley, R. (2021). Professional competencies in computing education: Pedagogies and assessment. *Proceedings of the 2021 Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 133–161).

Sánchez-Cifo, M. A., Bermejo, P., & Navarro, E. (2023). Devops: Is there a gap between education and industry? *Journal of Software: Evolution and Process*, *35*(12), e2534.

Salmon, P. M., Walker, G. H., M. Read, G. J., Goode, N., & Stanton, N. A. (2017). Fitting methods to paradigms: Are ergonomics methods fit for systems thinking? *Ergonomics*, *60*(2), 194–205.

Sharma, S. (2006). An exploratory study of chaos in human-machine system dynamics. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *36*(2), 319–326.

Silberstein, J., & Spivack, M. (2023). Applying systems thinking to education: Using the rise systems framework to diagnose education systems. *Research on Improving Systems of Education (RISE)*. doi: 10.35489/BSG-RISE-RI2023/051.

Spain, S. (2019). Systems thinking applied to curriculum and pedagogy: A review of the literature. *Curriculum Perspectives*, *39*, 135–145.

Sterman, J. (2018). System dynamics at sixty: The path forward. *System Dynamics Review*, 34(1–2), 5–47.

Syeed, M. M., Shihavuddin, A., Uddin, M. F., Hasan, M., & Khan, R. H. (2022). Outcome based education (obe): Defining the process and practice for engineering education. *IEEE Access*, *10*, 119170–119192.

Verma, R., Verma, S., & Abhishek, K. (2024). *Research methodology*. Chhattisgarh: Booksclinic Publishing.

Videnovik, M., Vold, T., Kiønig, L., Bogdanova, A. M., & Trajkovik, V. (2023). Game-based learning in computer science education: A scoping literature review. *International Journal of STEM Education*, *10*(54), 1–23. doi: 10.1186/s40594-023-00447-2.

Walker, G. H., Stanton, N. A., Salmon, P. M., Jenkins, D. P., & Rafferty, L. (2010). Translating concepts of complexity to the field of ergonomics. *Ergonomics*, *53*(10), 1175–1186.

Walker, G., Salmon, P., Bedinger, M., & Stanton, N. (2016). What the death star can tell us about ergonomics methods. *Theoretical Issues in Ergonomics Science*, *17*(4), 402–422.

Wiedemann, A., Wiesche, M., Gewald, H., & Krcmar, H. (2023). Integrating development and operations teams: A control approach for devops. *Information and Organization*, *33*(3), 100474.

Wilson, J. R. (2014). Fundamentals of systems ergonomics/human factors. *Applied Ergonomics*, *45*(1), 5–13.

Woods, D., & Dekker, S. (2000). Anticipating the effects of technological change: A new era of dynamics for human factors. *Theoretical Issues in Ergonomics Science*, *1*(3), 272–282.

Zarour, M., Alhammad, N., Alenezi, M., & Alsarayrah, K. (2020). DevOps process model adoption in saudi arabia: An empirical study. *Jordanian Journal of Computers and Information Technology*, *6*(3).

Zhong, F., Huang, S., & Lin, Y. (2023). The application of "organ system-based learning" digestive system teaching model in the clinical internship teaching of undergraduate nursing students. *Chinese Journal of Medical Education Research*, *22*, 1246–1251.