



DevOps Approach Embraces Forward and Reverse Engineering

Acep Taryana^{a,*}, Ari Fadli^b, Eko Murdyantoro^c, Siti Rahmah Nurshiami^d

^{a,b,c} Department of Electrical Engineering, Universitas Jenderal Soedirman, Indonesia

^d Department of Mathematics, Universitas Jenderal Soedirman, Indonesia

acep@unsoed.ac.id, arifadli@unsoed.ac.id, eko.atmojo@unsoed.ac.id, siti.nurshiami@unsoed.ac.id

ARTICLE INFO

Received June 5th, 2020
Revised September 21st, 2020
Accepted September 22nd, 2020
Available online May 24th, 2021

Keywords

Automated Tools, DevOps, Object-oriented Development, ORM, Software Engineering.

ABSTRACT

Modern software development methods such as Agile have given customers a flexibility to provide new requirement input when development is on progress, but customers cannot access released products during development. Nowadays, DevOps is a new method of software development that is the solution to these problems. In general, the DevOps Method does not emphasize complete system requirements at the beginning of development. Instead, the formulated requirements are immediately drafted in the model, implemented, and deployed, so the customer quickly obtains an overview of the product. This paper aims to discuss forward and reverse engineering software development in DevOps infrastructure. This paper is limited to the discussion side of software development engineering, it does not discuss the side of daily operations such as the discussion of web servers and other subsequent processes. Through a case of developing an internal quality assurance system at UNSOED, it was shown that forward and reverse engineering did not affect the stability of software development and operation using the DevOps method. The results of the study show that forward and reverse engineering are parts of development phase, be done because of the existence of new requirements from customers or improvement from developer itself, be done concurrently with the operation phase.

We would like to thank the DevOps partners in Universitas Jenderal Soedirman, especially the LPTSI and LP3M institution, so the DevOps research could be conducted.

* Corresponding author at:

Department of Electrical Engineering, Universitas Jenderal Soedirman, Indonesia

Jl. Mayjen Sungkono KM. 5 Blater, Kabupaten Purbalingga, Jawa Tengah 53371 Ph. +62 (0281) 6596700 Fax. +62 (0281) 6596801

Indonesia

E-mail address: acep@unsoed.ac.id

ORCID ID:

First Author: 0000-0001-9302-3540

<https://doi.org/10.25124/ijait.v4i02.2865>

Paper_reg_number IJAIT000040211 2020 © The Authors. Published by School of Applied Science, Telkom University.

This is an open access article under the CC BY-NC 4.0 license (<https://creativecommons.org/licenses/by-nc/4.0/>)

1. Introduction

Nowadays, the issue of DevOps is fascinating to study because it is not just methodologies. However, it is philosophically equipped with tools to run automation, collaboration, continuous integration, continuous delivery played by business teams, software development teams, and operations teams in organizational structures that are more complex and difficult to navigate [1]. DevOps has become a solution for both organizations that act as developers and operations that have the nature or character of work that is contradictory to the development and operation of the system. The development team tends to have a dynamic character in system development. In contrast, the operating team is more likely to maintain the infrastructure's stability so that the application runs. DevOps' use will increase the effectiveness of work culture from the development and operation of the system [2], which will improve collaboration between the development and operations team. Although natural conflicts occur between the two different organizations, they must be in harmony with each other to carry out the development and operation cycle [3]. Another issue, DevOps can identify and reduce the gap between the Development and the Operations team. Professional software developers often tend to develop views where product changes are what they are mandated to accomplish [4] [5].

Research on software development phases has been prevalent, but research related to its effect on the DevOps approach is still rare, few, even less. This research is fundamental because the DevOps approach can demonstrate software development outcomes according to customer needs through a monitor/terminal within a selected development infrastructure [6]. It is easy; customers can access or use the software whenever there is a need change, including minor changes such as display details. The DevOps approach can also collaborate with developers to produce software releases that will be deployed in operation environments accessible to customers.

This study aims to investigate the stages and details of applying the DevOps methodology in software development. We know that software development techniques can be through forward or reverse engineering. Something is interesting that should be observed. Could the two techniques be implemented in the DevOps methodology? How does both of technique be described in the DevOps? Then, what effect of shifting the selection of development paradigms from objects to non-objects when running both of these techniques in the implementation of DevOps.

2. Methods

It needs to be briefly discussed that object-oriented methodologies are chosen in this research. Object-oriented methods codify software engineering principles in terms of objects and class abstractions and determine processes, including concepts, mechanisms, and notations, to support them. Objects in this methodology can be compared with components in hardware design [7]. However, based on researchers' observations, the availability of developers who can implement the thinking of analysis and object design into object-oriented programming, even to the implementation of object-oriented databases, is still limited. Therefore the researcher designed this research using two approaches, an object-oriented approach for analysis and design phase, then switch to a mixed approach when designing a database. The changes in the use of the paradigm shift were observed, how much influence on the DevOps.

2.1. Best Practices of the Forward and Reverse Engineering

Forward Engineering starts from the work of analysis, design, and coding. While Reverse Engineering is a development work originating from the source code, then design, and analysis [8]. As a simple example, when this initial study was made in 2017, researchers compiled software through forwarding Engineering. Furthermore, this research was carried out by integrating forward and reverse engineering [6]. Reverse engineering occurs because of the addition or change of the source code without going through the analysis and design phases. Reverse engineering activity is identical to knowing the old source code without having the complete design documentation [9]. Reverse Engineering has been needed to measure existing systems and reconstruct the model according to the original [10]

Forward and reverse engineering activities are also identical with trial and error development activities. Although trial and error, it has an impact on the speed of software development. Because each phase of work can be re-evaluated through reverse engineering after getting additional or modification, work results at a particular phase can be traced, remodeled into the work in the previous phase. It is almost certain that there is no trial and error in the waterfall methodology because the waterfall is oriented towards finishing products with complete input requirements [11]. Whereas in agile methodology, trial and error is an activity that accompanies throughout the development cycle, coincides with the existence of reverse engineering techniques in software development to obtain products that meet user needs [11]. The agile approach explains that programming activities begin when the design has not been completed, even testing the program can be started since the analysis activity is still not finished. This shows that the development phase activities have no strict limits so that it is possible to carry out trial and error.

2.2. How does the Forward and Reverse Engineering Work in the DevOps Approach?

Currently, DevOps loads explicitly about the phases of planning, coding, debugging, testing, deployment, operating, monitoring [12]. However, the phases of analysis and modeling have not been accommodated. This may be the case because the analysis and design phases have not been entirely quantifiable. Someday, DevOps allows writing by including phases of analysis and model design, including considering computational model accuracy.

DevOps' role in software engineering is to connect all development disciplines at various phases into the technological infrastructure of software product development and operation [12]. Through DevOps, the gap between thinking and implementation is reduced, and the gap between customer and developer is reduced. This means that what the developer thinks will be quickly implemented. The customer can access the results of his thoughts. Moreover, what the customer wants is quickly realized and quickly known, re-evaluated by the customer.

Table 1 shows that the DevOps approach is closer to the agile approach. Some researchers categorize that DevOps is an approach to implementing the Agile methodology [13]. The DevOps is an extension of agile with the addition of product automation to improve collaboration between development and operation teams [14]. Because the agile approach's nature is to serve fast customer demands, rapid software development is needed not only forward engineering but also reverse engineering.

Forward engineering activities discuss how thought is expressed in a design model that can be arranged in a skeleton code. In contrast, reverse engineering

discusses how source code can be modeled in model design. Then what if using the DevOps methodology, does it accommodate forward and reverse engineering? Note that DevOps based on Table 1 is an extension of agile methodology. The difference is, Agile only covers the discussion of "Development" while DevOps, not only "Development" but also discusses "Operation." Based on the explanation of the DevOps cycle, the Development stage is equal to half the work of all DevOps work, and so is the operation phase [15][4]. The accumulation of all Development and Operation jobs will form a single DevOps cycle.

Table 1 The Software Development Approach Describes the Product Issue, Cycles, and Automation

Approach	Properties		
	Product Issue	Cycles	Automation
Waterfall	Finish product	Complete requirements are clear and fixed	Product might only be working on a developer's computer/in a test environment
Agile	Iterative and incremental	<ul style="list-style-type: none"> • Requirements change frequently • Development needs to be agile 	Product might only be working on a developer's computer/ in a test environment
DevOps	Iterative and incremental	<ul style="list-style-type: none"> • Requirements change frequently • Development needs to be agile • Operation needs to be agile 	Automatically the product not only be working on developer's computer but it is properly working upon operation infrastructure inside the customer environment

3. Results

Based on the software development cycle, the DevOps approach starts working gradually from coding to implementation. The automation of phases is done by several tools used to support the implementation of DevOps. The tools are grouped into two parts. The first is a tool whose role is to organize, collaborate in making source code. The second is a tool whose role is to install configurations, deploy applications on the server automatically. The second tool can access the source code managed by the first tool. Updating the source code every time will trigger a second tool to perform configuration updates applications and implement applications. The explanation of both is discussed below.

3.1. From the Object-oriented Technology into a Relational Database

Before the DevOps approach was implemented, programmers are assigned to design ER databases as initial deliverables, which were solutions to a business information system process's problems. Of course, the working model is very magical. Programmers directly translate the requirements into database designs. If a development model like that continues to be maintained, it affects development productivity measurement. The DevOps approach provides automation throughout the software development phase, not only in the programming phase but also in the analysis and design, testing, and deployment phases.

In this study, object-oriented technology (OO) is chosen at the analysis, design, and coding level. However, the use of technology is shifted during database implementation, using the concept of database relational. This adapts to the field conditions for the availability of programmers and tools. The development tools used are Visual Paradigm version 14.2 (VP). The VP uses UML notation in which this research uses Use Case diagrams, Class Diagrams, Sequence diagrams. The programming tool used is Eclipse with the programming language chosen is the Play Framework version 1.5. From object-oriented to non-object, the development paradigm shift is not a problem because it is supported by the existence of a connecting layer between persistence objects and relational databases called Object Relational Mapping (ORM). The ORM is the main need to build software with an

object-oriented approach, while the database used is already a relational database. With the ORM, programmers work using object concepts, and then tools will translate automatically into the concept of relational information in the database. The relational object mapping (ORM) framework overcomes the problem of the impedance level of software implementation.

Table 2 Relation between OO Paradigm and Relational Database through all Phases of DevOps Approach

Phases	Paradigm	Database Implementation
Analysis	OO	-
Design	OO	-
Programming	OO	OO or Relational
Testing	OO	-

Table 2 shows that there are options for implementing databases, namely using object databases or relational databases on development models using an object-oriented paradigm.

3.2. Case Study for One Software Development Cycle

This section shows software development through the phases of analysis and design, implementation, deployment, which is built through DevOps infrastructure.

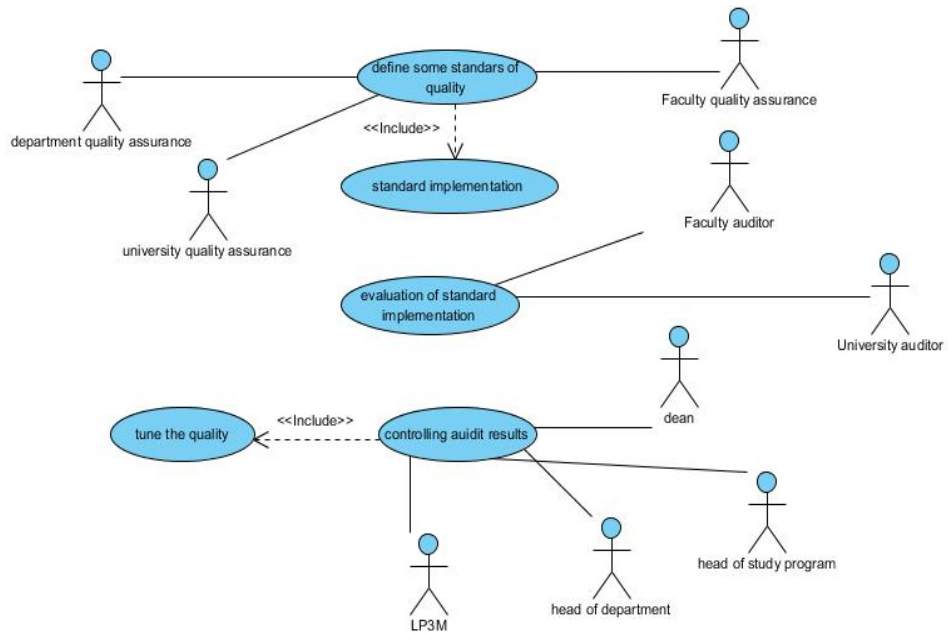


Figure 1 A Use Case of Internal Quality Assurance System

SPMI-PT application is one project which is used as a case for developing the University's internal quality assurance system at Universitas Jenderal Soedirman (UNSOED). The next section explains the phases of developing the SPMI-PT application using DevOps technique and tool. First, Figure-1 describes the scope of an internal quality assurance system for UNSOED. The figure is a model element from UML notation known as Use Case Diagram. The actors involved in quality assurance include department quality assurance, faculty quality assurance, university quality assurance, faculty auditors, university auditors, head of study programs, head of the department, dean, and Quality Assurance Agency, namely

LP3M. Based on the picture, all actors have the right to access (create/read/update/delete) information through the corresponding use case. For the first example, an actor of department quality assurance, faculty quality assurance, and university quality assurance could define some standards. The defined standard is the targeted quality to be achieved on a measurement. The second example, actors of faculty auditor, university auditor, could conduct to evaluate implementation. The evaluation phase can be executed after the standard definition, and implementation phase has been completed. The last example is actors of dean, head of the study program, LP3M, head of the department could access the controlling audit results use a case in which the actor can then access the tune quality use case. As the last phase of the implementation of quality assurance in each cycle, controlling and tune use case is a use case that is very important to be able to improve the quality that has been targeted, whether the quality is achieved or not. The results of the quality control can be upgraded or adjusted to the real conditions in the field.

The picture shows one relationship, including between "define a standard" and "standard implementation" use case; another include relationship is between "controlling audits" and "tune quality" use case. The relationship gives the meaning that the use case is a continuation of the related use case. Second, this step is to create a class diagram representing an information structure of the SPMI-PT that is built. The class diagram is a diagram as a form of the first release of the SPMI-PT application, where there are still many structural limitations. The most important is the first release application that can describe users' needs of the quality assurance system. The class diagram illustrates the presence of standards structure consists of a quality assurance unit structure, quality assurance structure, and quality indicator structure. The class diagram in Figure 2 describes the relationships between classes, such as the standard relation between unit, the relation between the responsible person, and the standard.

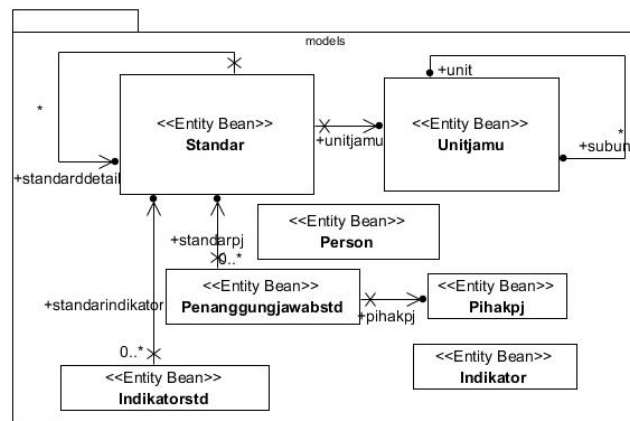


Figure 2 Class Diagram of Internal Quality Assurance System at Universitas Jenderal Soedirman [6]

Figure 1 and Figure 2 are products from the analysis and design phases, respectively. To make these products, we use the VP. This phase in the DevOps approach is not explicitly stated. However, based on literature [15], it is part of the "Plan" of the DevOps approach. More detail Plan phase includes capture requirements, create plans and tasks, analysis, scoping.

Third, it is an implementation phase in the DevOps known as the "Coding" phase. To realize the coding phase, we first generated the class diagram in Figure 2 using the VP tool. The result is a skeleton code that must be detailed into the full

source code. The skeleton code generated is standar.java, unitjamu.java, person.java, penanggungjawabstd.java, pihakpj.java, indikatorstd.java, indikator.java. Source Code 1 is an example of a source code snapshot that has been created using an Eclipse editor implemented in Play Framework 1.5 (PF15).

Source Code 1 Realization of the Standard Class (standar.java)

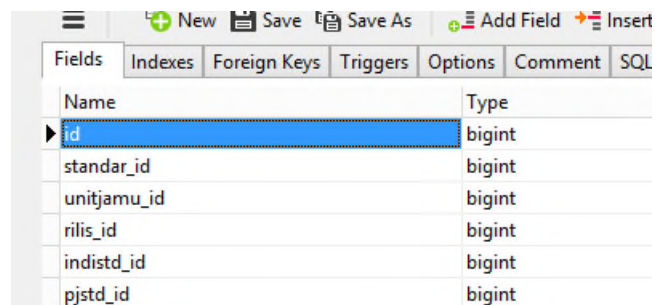
```

42 @OneToMany(mappedBy="upstandar", cascade=CascadeType.ALL)
43     public List<Standar> standarddetail;
44
45 @ManyToOne
46     public Standar upstandar;
47
48 // @OneToMany(mappedBy="pj", cascade=CascadeType.ALL)
49 //     public List<Pihakpj> pj;
50
51 @OneToOne
52 @Required
53     public Unitjamu unitjamu;
54
55 @OneToMany(mappedBy="standarpj", cascade=CascadeType.ALL)
56     public List<Penanggungjawabstd> pjstdr;
57
58 @OneToMany(mappedBy="standarindikator", cascade=CascadeType.ALL)
59     public List<Indikatorstd> indikatorstdr;
60
61

```

The focus of this phase is how to complete the skeleton code into a complete code that minimalist. The minimalist limits indicate that source code is considered sufficient if it does not show syntax errors when compiled. A question arises, does all the generated skeleton code have to be complete? The answer adjusts the minimum threshold conditions. It should be borne in mind that we demonstrate the superiority of a DevOps approach where software development starts when some small ideas directly implement and are executed by computers so that the application functions appear to run in a DevOps environment.

The development of an application using PF15 requires a model-view-controller (MVC) as an architectural pattern. By default, if creating a project using PF15, the MVC folder will be created. We have only a minimalist Model (M) that contains a complete skeleton code until this phase. The next target is that the application must be equipped with a database that contains the entity of the system being built. In this case, the database structure can be generated from the model compiled in the PF15 automatically. The result of the compilation is shown in Figure 3. This is one of the advantages of enterprise application development using an object-oriented approach with the help of PF15. The impact, a developer, is not busy with database design work.



Name	Type
id	bigint
standar_id	bigint
unitjamu_id	bigint
rilis_id	bigint
indistd_id	bigint
pjstd_id	bigint

Figure 3 Fields of Standard Table or Entity

According to the DevOps approach, the step to compile and create a source code that can be executed is a Build phase. To support the work of the Build phase, it takes two tools. The first is a tool for collaborating and integrating code. The second is a tool for compiling and deploying to be an executable program in the prepared environment. Through using GitHub and Jenkins tools, a developer could carry out coding and simultaneously to collaborate management and deploy applications automatically. In the example above, the developer has created the source code using the PF15, using GitHub to implement collaboration management, and using Jenkins to conduct continuous integration [6]. We could show the work process in recording every condition of change in collaborative management tools such as GitHub and Jenkins as in the following Figure 4 and Figure 5, respectively.

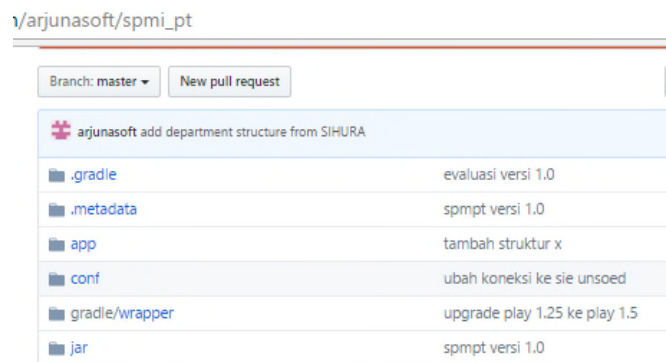


Figure 4 Management Collaboration Using GitHub at SPMI-PT Application

The information in Figure 5 shows that the SPMI-PT application could already be run on the target server. The application users can already access it through interfaces that have been deployed into one server, such as the view in Figure 6. At the same time, they could evaluate if there is an incompatibility with the requested requirements.

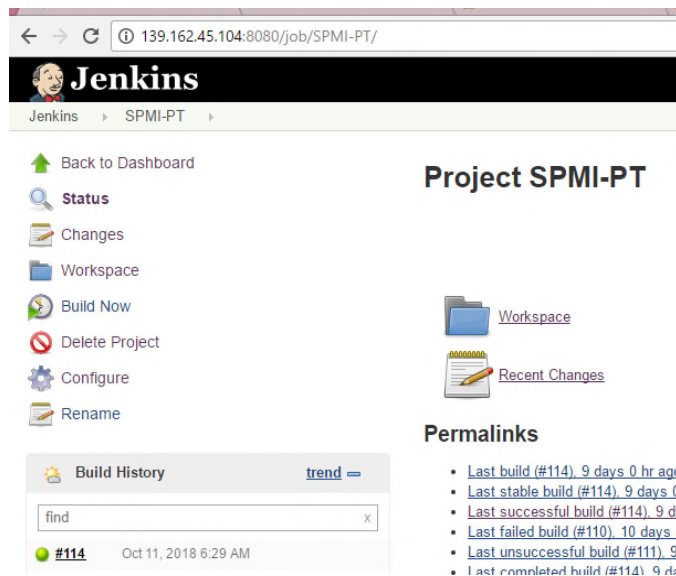


Figure 5 Continuous Integration Using Jenkins for SPMI-PT Application

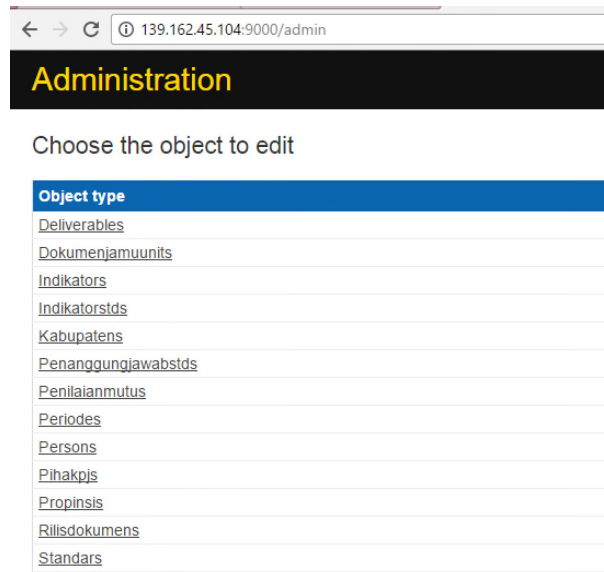


Figure 6 Default CRUD for SPMI-PT Application

Source Code 1, Figure 1 Figure 2 Figure 3 Figure 4, Figure 5, Figure 6 are products generated from the Forward Engineering phase if it is done sequentially. What about the Reverse Engineering phases that developers cannot avoid when it comes to a new requirement into the system being developed? As an appropriate solution, the VP can directly facilitate reverse engineering from source code change into the design phase. This section describes the change in a code that can affect the structure of the database, and of course, the deliverables at the design phase must be changed. For example, a developer creates a Source Code 2 without a thorough analysis and design phase.

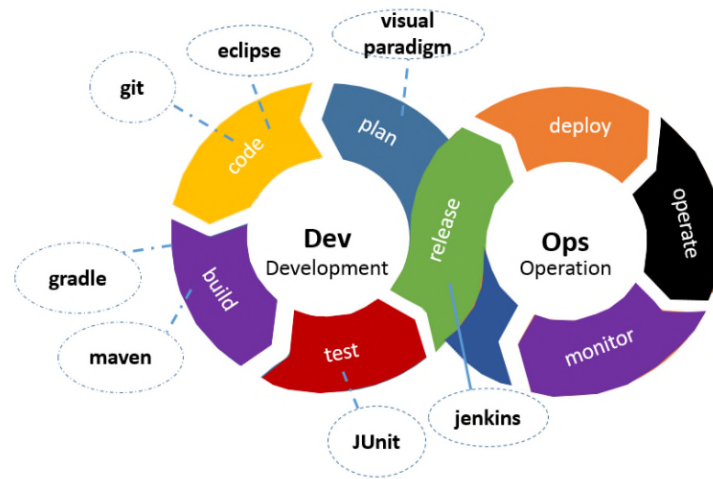


Figure 7 DevOps Cycle and Related Tools

In agile methodology, it is permissible and legitimate. And then, how with DevOps approach? Section 2 discussed that forward and reverse engineering is an activity half part of the DevOps cycle. Half of the DevOps cycle on the development side is identical to agile methodology. Therefore, DevOps can do that as well in agile. The DevOps cycle and accompanying tools can be seen in Figure 7.

Figure 8 shows the direction of software development in DevOps can be through forwarding Engineering or Reverse Engineering. If the Forward Engineering direction is taken, the Plan, Code, Build, Testing, and Monitor columns must be done in sequence. However, if the Reverse Engineering direction is taken, the reverse direction from forwarding Engineering is taken, analysis and design are done after working on the source code. For example, in the direction of forwarding Engineering, the first step composes the analysis, the second step arranges the design. The analysis results are shown in Figure-1, and the design results are shown in Figure 2. Steps 1 and 2 are part of the Plan in DevOps. The next step is the 3rd step to make the source code based on the 2nd step results. Still, in the 3rd step, the source code is set in the GitHub. The 3rd step is part of the Code in DevOps. In the 4th step, successively set the source code, carried out the build using Jenkins, and executed the generated database when the application was successfully executed. Still, in that stage, the application is deployed on the destination server. The 4th step has been carried out sequentially by the Build and Deploy phase. In the last step in forwarding Engineering, the customer can see the software product developed through the specified server interface. Steps 1, 2, 3, and 4 above are defined as the first iterations in half of the DevOps cycle.

Meanwhile, reverse engineering is described as follows. When the source code has been changed, the designer can reverse engineering it into design and analysis. Changing the source code is done in the 3rd step, while generate design is carried out in the 2nd step in the Plan phase. When reverse engineering is done, it is possible that the product has been released and delivered to the customer. Reverse engineering is done because there is a source code repaired by the programmer that needs to be adjusted for a new design or a new requirement of the customer that is done on the next iteration of the DevOps cycle.

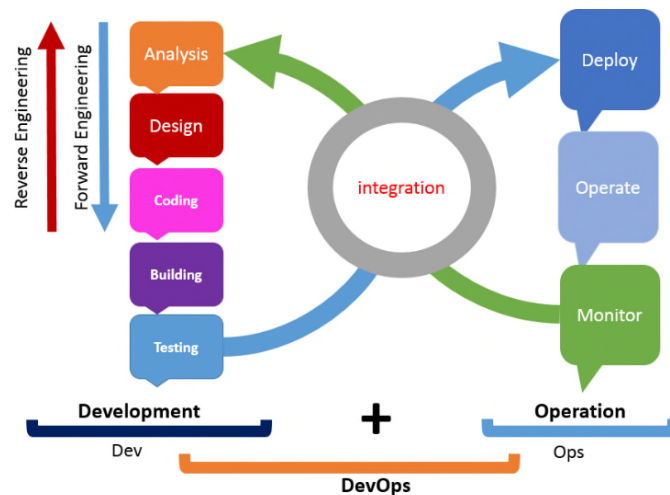


Figure 8.DevOps Cycle in other Form

Reverse engineering can be done during software development running before the product is released, delivered to the customer, or done when the product has been delivered. The customer has used it through new requirements. Figure 9 shows the behavior of reverse engineering work on the development side. The figure also shows that reverse engineering work can be done simultaneously with the operations section.

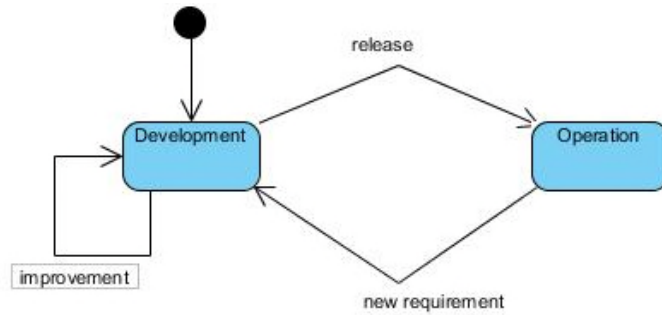


Figure 9 DevOps State Machine

Similarly, with the steps in the Forward Engineering approach, Source Code 2 is compiled and then pushed to GitHub if it is free of errors. Jenkins will automatically execute the build command and deploy it to the target server. After the Jenkins implements build and deploy, the application and database are automatically formed immediately. This step is the last in one cycle before the customer can see changes in the deployment environment.

Source Code 2 New Class which represent Department Entity (department.java)

```

1 package models;
2 import java.util.*;
7
8 @Entity
9 @Table(name="department")
10 public class Department extends Model {
11
12     @ManyToOne
13     @Required
14     public Company upcompany;
15
16     @OneToMany(mappedBy="atasanlangsungdepartment", cascade=Cascade
17     public List<Department> listpembantu;
18
19     @OneToMany(mappedBy="updepartment", cascade=CascadeType.ALL)
20     public List<Department> listdepartment;
21
22

```

The next step that must be done when implementing reverse engineering is generating Class diagrams from the source code. This step includes part of the automation in the DevOps approach. This is an important step in reviewing the product through class design that represents the source code's skeleton. The structure can be used by designers to evaluate suitability. Figure 10 is a reverse result of the source code being a class diagram.

Finally, in reverse engineering, we must add, update, delete some models in the use case according to the newly formed class diagram. The class diagrams are created automatically using VP, while the use case diagrams are arranged by adding additional parts manually.

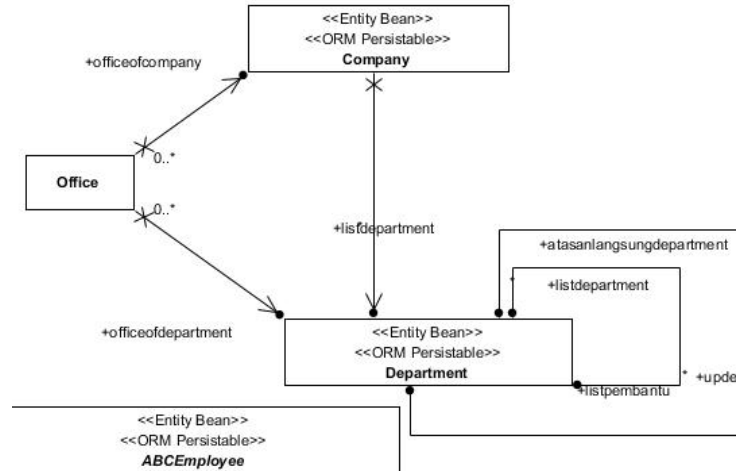


Figure 10 New Class diagram Created After the Addition of A New Code in the Reverse Engineering Approach. (department.java into department class diagram).

3.3. Forward and Reverse Engineering on Extreme Programming (XP)

XP is one of several popular Agile processes, apart from Scrum, Crystal Clear, Adaptive Software Development, Feature Driven Development, and Dynamics Systems Development Method (DSDM). Because Agile is part of SDLC, XP also has various work stages, which generally include Planning, Design, Coding, Testing.

It is essential, XP is applied to work projects where the initial definition is not clear; the customer does not have a clear initial picture of the output [16]. Therefore, it is very appropriate that software development work is carried out in a forward and reverse direction that is fast, user-oriented, and anticipates, rapidly changing user needs. Forward and reverse engineering is carried out so that users can see the desired product quickly. Even when there are mismatches, the user can provide new input to the developer quickly.

XP was a fast method widely used in software development at its time. The characteristic of XP that most influences the quality and speed of software deployment is pair programming. With pair programming, programmers work on one computer to complete the program. What if the programmer is more than two people and scattered everywhere? XP cannot address these issues.

Based on this discussion, it can be said that XP is a complementary part of the DevOps method. Both are agile software development approaches or methods. Nevertheless, based on the previous sub-section explanation, DevOps is not just a method or approach. It also discusses tools and infrastructure as a place or tool to run software development. More importantly, the scope of it discusses the automation of the development stage and the operation stage. It is a solution to reducing the gap between developers and operators, where the development team tends to radical changes while the operating team tends to be stable. Furthermore, it discusses developer collaborations of more than two people in various places simultaneously working together [6].

4. Conclusions

We have demonstrated a series of software application development using the DevOps approach. The development model was chosen in two ways, namely forward and reverse engineering. The result is that both forward and reverse

engineering can be run at various DevOps cycle stages, including Plan, Code, Build, Test, and Deploy. Automation of the implementation of forwarding and reverse engineering has been recorded in half part of one DevOps cycle, namely in the side, including Plan, Code, Build, Test, Deploy. Execution of forwarding and reverse engineering activities in the DevOps approach shows that each job in a stage can be traced. The traceability is a requirement so that continuous development can simultaneously run from analysis to testing or vice versa.

Forward and reverse engineering is part of the Development (Dev) stage that is done because of the improvement of the source code from the developer itself or because of new customer requirements after the operation (Ops). After one DevOps cycle is reached, development work can be done simultaneously with the operation, known as concurrent tasks. Every time a new change is triggered by the developer or from the customer, the next iteration of the DevOps cycle will be launched, and a new release will be launched soon.

Our study results also show another aspect, namely the stability of the software development process in the DevOps approach. Software development that begins with an object-oriented approach must shift software development that begins with an object-oriented approach that must shift into a non-object approach when it implements a database. With current technological advancements, this paradigm shift can be solved using a programming framework with the ORM concept. By using the ORM concept, developers are not preoccupied with database design, but more effort is devoted to compiling a class diagram model to solve the problem. During execution, ORM is a layer that functions as a bridge between the model class and database implementation.

It is quite interesting to do it in the future to quantify the quality of software development built with the DevOps approach. As an opening discussion, this study has revealed the aspects of Culture, Process, and Technology as some supporting components of the development process. The description of the critical work stages has been described in half of the cycle of the DevOps. Note that half of the DevOps cycle is an agile methodology. This shows, in general, that developing software using DevOps will be better than previous methodologies such as agile.

Bibliography

- [1] J. M. Jez Humble, "Why Enterprises Must Adopt Devops to Enable Continuous Delivery," *Cut. IT J.*, vol. 24, no. 8, 2011.
- [2] H. Niwa, "Towards the adoption of DevOps in software product organizations: A maturity model approach," *Development*, vol. 134, no. 4, pp. 635–646, 2007, DOI: 10.16242/j.cnki.umst.2014.04.005.
- [3] O. Reilly, "Michael Hüttermann," in *Beginning DevOps for Developers*, 2012, pp. 3–13.
- [4] J. Wettinger, U. Breitenbücher, and F. Leymann, "DevOpsSlang - Bridging the gap between development and operations," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8745 LNCS, pp. 108–122, 2014, DOI: 10.1007/978-3-662-44879-3_8.
- [5] B. S. Farroha and D. L. Farroha, "A Framework for Managing Mission Needs, Compliance, and Trust in the DevOps Environment," in *MILCOM '14 Proceedings of the 2014 IEEE Military Communications Conference*, 2014, pp. 288–293, [Online]. Available: <http://dx.doi.org/10.1109/MILCOM.2014.54>.
- [6] A. Taryana, I. Setiawan, A. Fadli, and E. Murdyantoro, "Pioneering the automation of internal quality assurance system of higher education (IQAS-HE) using DevOps approach," <https://ieeexplore.ieee.org/document/8304146>, 2018, DOI: 10.1109/SIET.2017.8304146.
- [7] E. Colbert, "Choosing the Right Object-Oriented Method," <https://www.researchgate.net/publication/228802941>, no. 760, 1992.
- [8] E. Chikofsky and J. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, 1990, [Online]. Available:

- <https://ieeexplore.ieee.org/document/43044>.
- [9] R. Singh, "A Review of Reverse Engineering Theories and Tools," *Int. J. Eng. Sci. Invent.*, vol. 2, no. 1, pp. 35–38, 2013, [Online]. Available: [http://www.ijesi.org/papers/Vol\(2\)1/G213538.pdf](http://www.ijesi.org/papers/Vol(2)1/G213538.pdf).
 - [10] C. Baidada, E. M. Bouziane, and A. Jakimi, "An Analysis and New Methodology for Reverse Engineering of UML Behavioral," *Int. J. Adv. Eng. Manag. Sci.*, vol. 2, no. 7, pp. 1012–1016, 2016.
 - [11] Abhinav, M. Vijayalakshmi, A. Bhandiwad, K. Mellikeri, and P. Nagesh, "Transition from Conventional to Agile Process Model an Experience Report," *J. Eng. Educ. Transform.*, vol. 0, no. 0, 2018, [Online]. Available: <http://www.journal.eet.org/index.php/jeet/article/view/120893/82987>.
 - [12] F. Erich, C. Amrit, and M. Daneva, "Report: DevOps Literature Review," https://www.researchgate.net/publication/267330992_Report_DevOps_Literature_Review, no. October, pp. 1–27, 2014, DOI: 10.1007/978-3-319-13835-0.
 - [13] A. Robert, T ; Masters, William ; Stark, "Teaching Agile Development with DevOps in a Software Engineering and Database Technologies Practicum," in 3rd International Conference on Higher Education Advances, 2017, pp. 1353–1362.
 - [14] P. Perera, R. Silva, and I. Perera, "Improve software quality through practicing DevOps," <https://ieeexplore.ieee.org/document/8257807>, no. March, pp. 1–6, 2017, DOI: 10.1109/ICTER.2017.8257807.
 - [15] S. Carrizo, S. Cucu, and S. Modir Using Liberty for DevOps, Continuous Delivery, and Deployment. 2015.
 - [16] R. Fojtik, "Extreme programming in the development of specific software," *Procedia Comput. Sci.*, vol. 3, pp. 1464–1468, 2011, DOI: 10.1016/j.procs.2011.01.032.