Implementation of DevOps based Hybrid Model for Project Management and Deployment using Jenkins Automation Tool with Plugins

Poonam Narang†, and Pooja Mittal†
poonammdu.rs.dcsa@mdurohtak.ac.in, pooja@mdurohtak.ac.in
†Department of Computer Science and Applications, Maharshi Dayanand University, Rohtak, Haryana, India

Summary
Project management and deployment has gone through a long journey from traditional and agile to continuous integration, continuous deployment and continuous monitoring. Software industry benefited with the latest buzzword in the development process, DevOps that not only escalates software productivity but at the same time enhances software quality. But the implementation and assessment of DevOps practices is expository as there are no guidelines to assess and improvise DevOps application in software industries. Hence, there was a need to develop a hybrid model to assist software practitioners in DevOps implementation. The intention behind this paper is to implement the already proposed DevOps hybrid model using suggested tool chains including Jenkins, Selenium, GitLab, Ansible and Nagios automation tools through Jenkins project management environment and plugins. To achieve this implementation objective, a java application is developed with a web-based graphical interface. Further, in this paper, different challenges and benefits of Jenkins implementation shall also be outlined. The paper also presents the effectiveness of DevOps based Model implementation in software organizations. The impact of considering other automation tools and models can also be considered as a part of further research.

Keywords:
Automation, Automation Tools, DevOps, Project Management, Software development

1. Introduction

Successful software development is always a major concern for every organization. Traditional software development methodologies suffered from many failures like late or run-away projects, risk mitigation, discontented customers and much more. DevOps being an emerging and new software engineering paradigm is adopted by different software organizations to develop within schedule and within budget quality software. In spite of so many benefits, implementation of DevOps culture suffers from different challenges and issues like the existence of lots of alternative automation tools and hence their accurate selection along with lack of performance measurement. To overcome these problems or issues, many tool chains or hybrid models were proposed. These models in the form of Integrated Tool Chain (ITC) not only accelerate the development process of software but also speed up the delivery process up to much greater extent. [1] These models or process improving techniques are also not similar for each and every organization so selection of appropriate model or automation tool becomes of utmost importance. This initial idea has been published already as a hybrid automated model from alternative DevOps automation tools for each stage of Continuous Integration, Continuous Testing, Continuous Delivery, Continuous Deployment and Continuous Monitoring. Our hybrid automated model [2], attempts to implement a "best of breed" solution with performance evaluative analytical comparisons of alternative automation tools available at different stages of software development. In this paper, we extend the work by implementing a hybrid model for project management and deployment using their best performer tools. Therefore, the prime objective of this paper is to develop an implementation model in DevOps culture through different tools like Jenkins, Selenium, GitLab, Ansible and Nagios. All these tools were incorporated as different plugins in Jenkins project management environment. To achieve this target, following steps have been developed and followed under this work –

(i) Installation of JDK for writing web based application in Java and that can be deployed to Tomcat
(ii) Setup of Jenkins, a DevOps continuous integration and build tool
(iii) Installing different required Plugins including suggested by hybrid automated model to extend the functionality of Jenkins
(iv) Final stage of CI/CD Pipeline is to deploy War file to application server Apache Tomcat

The rest of the paper is assembled as follows: Section 2 represents related background study. Next sections highlight already existing models, motivation behind the work, research design, and hybrid automation model followed by implementation through different alternative automation tools. Finally implications and findings along with conclusions and future work shall be discussed.

Manuscript received August 5, 2022
Manuscript revised August 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.8.31

2. Related Study

Project development and deployment also termed as Software Development Life cycle or SDLC involves standard procedure or process for formulation of software. SDLC lists different phases in the life cycle of software which are followed by development industries to deliver software products. SDLC has gone through many revisions from traditional and agile models to recent DevOps culture. Different existing and related renowned literature is reviewed and studied in terms of DevOps research papers.

2.1 Traditional, Agile and DevOps-an Overview

DevOps – Development and Operations is a recent, emerging paradigm in software evolution. It bridges the communication gap between development and operations teams and targets to reduce the discrepancies of different teams [3]. Traditional methodologies do not focus on these tasks explicitly. Lwakatare et al in their research on case study of five companies [4] also agrees for coordination between development and operations teams. As many renowned researchers restrict the adoption of DevOps in practice though there are multiples of theories that are against DevOps application and talk about its challenges and lack of performance measure [5]. For example, Leite, Roacha and others conducted a survey in their paper [6] and discussed different challenges in DevOps adoption. Other researches also force the compulsion of DevOps practices for the organization to move towards delivering higher performance and quality software [7].

Similarly, Ronny Olguin [8] and Ramtiin Jabbari et al [9] in their explicit papers on DevOps highlight that DevOps acts as a movement to automate the tasks of continuous delivery of new software updates while at the same time guaranteeing their correctness and reliability. Authors [9] also conducted systematic literature review on the definition of DevOps and agrees that DevOps extends the agility component in software development paradigm.

2.2 Existing Models of DevOps

The literature shows the development of several models for the guidance of industry or software practitioners for fruitful and successful implementation of DevOps in practice. Many models have been proposed in this context like the unicorn framework proposed by Trihinas and others [10], to overcome different challenges of DevOps through continuous releases. Similarly, other model DORA proposed by Forsgren et al [11] talks about successful product delivery and Syed W. Hussaini in DevOps paper [12] accepts the emerging DevOps paradigm as a response to the growing knowledge of the existing gap of 4 Cs (Communications, Cooperation, Culture and Collaboration) between development and operation teams functions of an organization. Authors also accept “Wall of Confusion” between these teams. This “Wall" is caused by a combination of conflicting motivations among people, processes and technology/tooling. Hence, the need for strengthening the harmonization of Dev and Ops teams arises. The model was also outlined in [12] for enhancing effectiveness and efficiency of DevOps stakeholders interest.

2.3 Motivation

Many DevOps models exist in literature and all talk about quality software delivery and development paradigm. Talks also include the existence of many alternative automation tools to achieve the targets of DevOps [13] but no model speaks about the difficulties in the accurate selection of these alternative automation tools. Therefore, it becomes of utmost importance to propose the inclusion or introduction to hybrid automated tool model [2] to automate the tasks of tool selection through Integrated Tool Chain (ITC) based on several performance evaluators. Our previous work involves the proposal of ITC and current underlying research work is the extension of [2] towards implementation. On the basis of existing models, their findings and by following their suggested road map, this paper implements the hybrid model.

3. Research Design

For this research, we have used Jenkins project management and deployment tools along with different tools of hybrid models like Selenium, GitLab, Ansible, and Nagios in the form of different Jenkins Plugins. A web based graphical interface as sample software application is also developed for the purpose of implementation. The details are as follows:

Phase 1:

(i) This step aims to develop the best web based graphical interface application for implementation in Jenkins Climate

(ii) A Java application with web-based graphical interface is developed and provided as Git repository in source code management

Phase 2:

(i) The aim of this step is to identify the hybrid model tools recommended in the tool chain and required environment for their proper installation as reported in the literature survey.

(ii) Setup of Jenkins Project Management tool from Java pipeline

(iii) Installing different required Plugins suggested by hybrid automated model to extend the functionality of Jenkins

Phase 3:

(i) The objective of this step is to evaluate quality parameters based on implementation outcome.

(ii) Finally the delivery pipeline is shown to include each build/release with verification of different metrics as quality parameters.

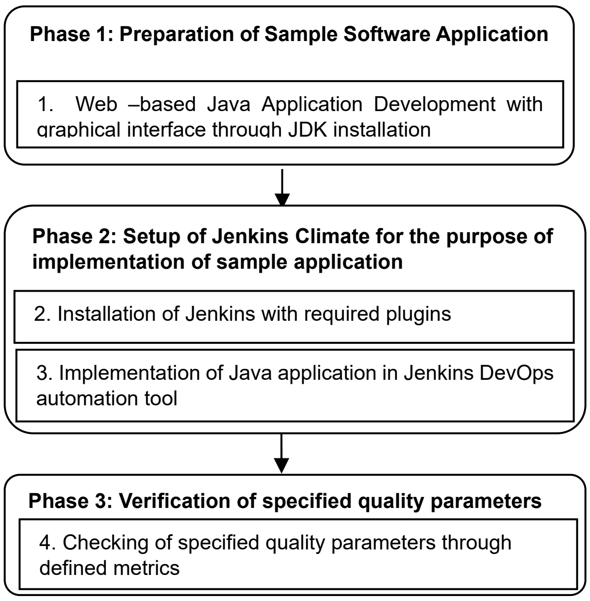

Phase 1: Preparation of Sample Software Application
1. Web –based Java Application Development with graphical interface through JDK installation
Phase 2: Setup of Jenkins Climate for the purpose of implementation of sample application
2. Installation of Jenkins with required plugins
3. Implementation of Java application in Jenkins DevOps automation tool
Phase 3: Verification of specified quality parameters
4. Checking of specified quality parameters through defined metrics

Figure1. Research methodology followed for the current research work

Figure (1) above includes diagrammatic representation of all phases and different steps to follow as per the methodology and also defined as research design.

4. Proposed DevOps based Hybrid Model

The proposed hybrid model paper [2] discussed DevOps culture in IT industries. It was found that DevOps provides complete automation in development and operations using different tools, and attempts to solve many industrial issues like delayed software releases, delivery or deployment to maintenance problems. In this work, performance evaluative comparison of different automation tools was done which further accelerated towards the design of an integrated tool chain (ITC). This tool chain of selective automation tools optimizes the performance of the delivery life cycle by removing different impediments at each stage. The ITC design in turn leads towards the evolution of DevOps based hybrid model of automation tools for software development. Following diagram clearly depicts the hybrid automated model consisting of these selective tool chains at different stages of DevOps culture –

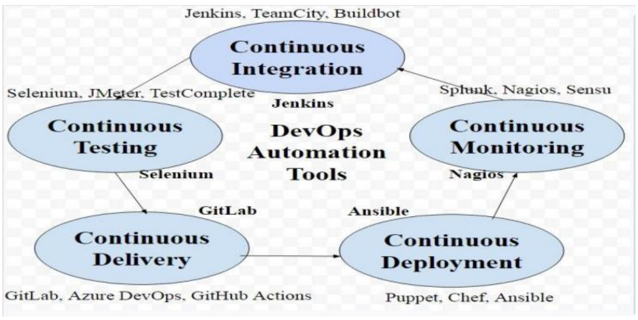


Figure2.DevOps based hybrid model of selective automation tools [2]

Figure (2) above, shows our proposed hybrid model of selective tools from an alternative set of DevOps automation tools. The underlying paper implements hybrid automation models to check the reliability of work for which it was not possible to consider already existing case studies. We have created a web based application under the JDK environment to evaluate an automation model that works as a data set for this paper.

5. Automation Tools Installed as Plugins In Jenkins

Multiple tools have been used from the development to the deployment of the project. These tools have been selected from Integrated Tool Chain (ITC) in our hybrid model for software development using DevOps automated tools [2]. Followed tools are explained below briefly –

(i) JDK1.2

Java Development Kit includes major features and enhancements to the Java platform. This is used to develop java based projects from the GitHub repository. Along with JDK, Tomcat Server is also used as a server to deploy java based applications. Tomcat server also supports continuous changes made to the project in terms of continuous integration by simply stopping and restarting the server.

(ii) Jenkins

Jenkins is a continuous integration tool that helps developers to deliver more predictable and reliable software. It restricts developers to integrate their updated code with the central repository periodically to maintain a more stable version of available code without any conflicts. Jenkins tool has been compared with other CI tools Teamcity and Bamboo. Based on different performance evaluators, a tabular comparison table has been shown in our work [2]. Current research has chosen Jenkins as a continuous integration or build tool with different plugins to support the development.

(iii) Selenium

Selenium has been selected as the best continuous testing tool with the scheduling of automation tests after every feature update performed by the developers. It overcomes many problems encountered in the traditional way of software testing and also ensures quality and best deployment on the other hand. A tabular comparison of Selenium has been made with other continuous test automation tools Jmeter and TestComplete on the basis of different performance evaluators and parameters [2]. Selenium has been installed as a separate Jenkins Plugin to automatically support test automation in an uninterrupted manner and also on a continuous basis.

(iv) GitLab

GitLab is selected as a continuous delivery tool that helps in automatic release and delivery of applications to reduce the deployment time. In hybrid model [2], GitLab is compared with other CD tools – Azure DevOps and GitHub Actions based on parametric table and market trend graph and concludes GitLab as the best continuous delivery tool. It supports automatic build creation of multiple code changes and also preparing for release or production. GitLab is also installed as Jenkins Plugins to incorporate its features into Jenkins itself.

(v) Ansible

Ansible is termed as the best Continuous deployment tool as chosen by hybrid automation model. Ansible allows software release process that is used for immediate autonomous deployment to the production environment after automated test validation. Continuous deployment offers remarkable benefits to modern software businesses. It also allows businesses to respond to teams along with meeting changing and increasing market demands to deploy and validate new features rapidly. [2]

(vi) Nagios

Nagios is chosen as the most commonly followed continuous monitoring tool in hybrid automation models as it enables faster and better response to changing needs of customers in contrast to traditional monitoring methods.

Current research work considers Jenkins, Selenium and GitLab automation tools along with Tomcat Server for the purpose of continuous deployment. Following section shows the usage of automation tools as implementation for the ToDoReact project.

6. Implementation of Model through Sample Application In Jenkins Environment

This paper follows three phased architecture to implement DevOps hybrid automation tool model. These three phases and different steps which are followed for the smooth implementation of the model are shown in the flowchart below –

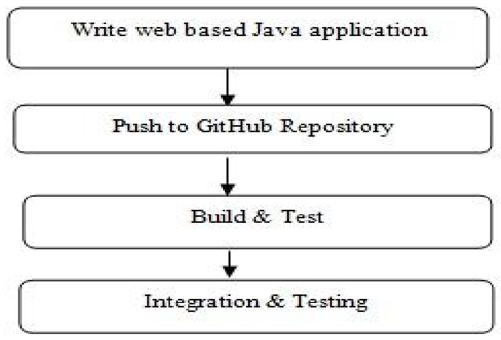
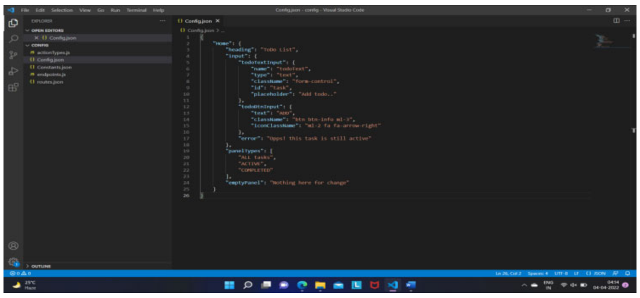


Figure3. Process flow of DevOps implementation approach

Jenkins being an Open-source CI/CD tool is used as a continuous integration and build tool as depicted through above figure (3). It allows continuous integration as well as continuous delivery of the software. Management of code is done through GitLab and GitHub repository. Different automation tools as in ITC [2] are installed as plugins of Jenkins. Selenium as Plugin for test case writing and execution is used. Ansible will take care of deployment, Nagios for managing operational procedures.

As shown in the above flowchart, the research approach begins with a sample java application written using a local repository and uploading the same on GitHub for which connection is already built with Jenkins. In Jenkins, a build is created using Maven that contains details of a java application. Selenium performs unit tests along with integration testing. Jenkins will perform continuous integration by itself followed by continuous delivery and deployment using different plugins without any manual intervention.The results of implementation of the system with detailed description is shown in screenshots below –

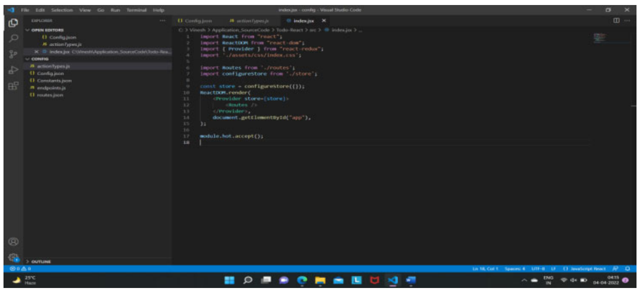
Step1. For installation of JDK and developing Java based web application named ToDoReact which provides user interface to add or delete new tasks to do or delete the completed tasks.



```
{
    "Home": {
        "heading": "ToDo List",
        "input": {
            "todoTextInput": {
                "name": "todoText",
                "type": "text",
                "className": "form-control",
                "id": "task",
                "placeholder": "Add todo..."
            },
            "todoBtnInput": {
                "text": "ADD",
                "className": "btn btn-info ml-3",
                "iconClassName": "ml-2 fa fa-arrow-right"
            },
            "error": "Opps! this task is still active"
        },
        "panelTypes": [
            "ALL tasks",
            "ACTIVE",
            "COMPLETED"
        ],
        "emptyPanel": "Nothing here for change"
    }
}
```

Figure4. Java Code developed in JDK environment for todo tasks

ToDoReact java web based code as shown in above figure (4) maintains a list of tasks to do along with the list of active and completed tasks.



```
import React from "react";
import ReactDOM from "react-dom";
import { Provider } from "react-redux";
import './assets/css/index.css';

import Routes from './routes';
import configureStore from './store';

const store = configureStore({});
ReactDOM.render(
    <Provider store={store}>
        <Routes />
    </Provider>,
    document.getElementById("app"),
);

module.hot.accept();
```

Figure5. Different status of tasks according to their being done or completed

These tasks' status can be updated according to incomplete /active /complete or it can be deleted from the list as depicted in above java code figure (5). The local structure of the code is shown in the figure (6) below –

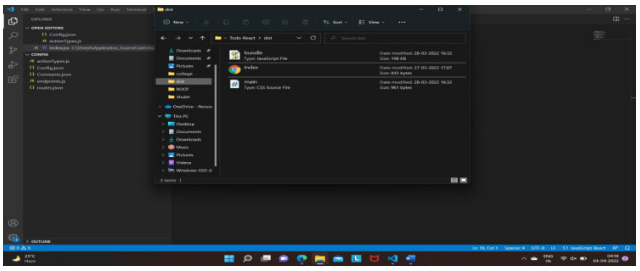
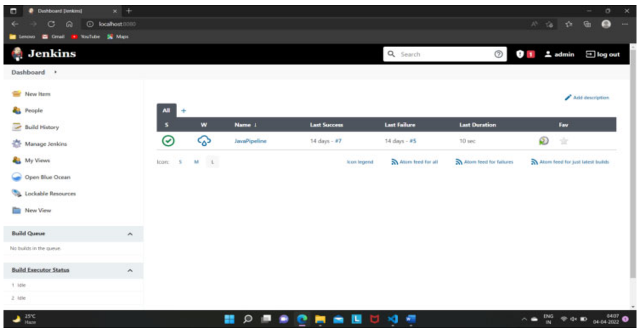


Figure6. ToDoReact Java application stored in system folder

After writing the code in the src directory of the application, the corresponding xml file is also updated.

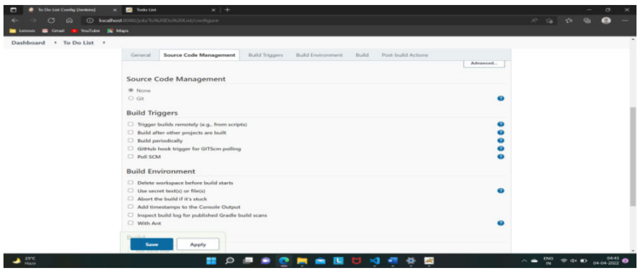
Step2. After completing java code next step includes the installation of Jenkins tool for continuous integration and build purpose as shown in figure (7) below –



S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
✓		JavaPipeline	14 days - #7	14 days - #5	10 sec	

Figure7 Jenkins automation tool for CI/CD

For Jenkins build job, it is needed to create a freestyle project type for which ToDoReact path is given as a repository address.



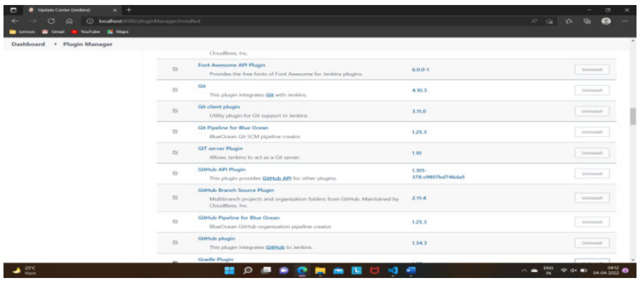
Source Code Management
- [✓ None]
- Git

Build Triggers
- Trigger builds remotely (e.g., from scripts)
- Build after other projects are built
- Build periodically
- GitHub hook trigger for GITScm polling
- Poll SCM

Build Environment
- Delete workspace before build starts
- Use secret text(s) or file(s)
- Abort the build if it's stuck
- Add timestamps to the Console Output
- Inspect build log for published Gradle build scans
- With Ant

Figure8 Building a freestyle project in Jenkins and configuring different environments

Figure (8) above shows the development of a Java project and for building java code in Jenkins it also requires inclusion of different plugin tools as given by hybrid model in [2]. Many of these plugins named – Ansible, Selenium, Nagios are already installed and GitLab is installed as shown in following figure (9) below –



Font Awesome API Plugin Provides the free fonts of Font Awesome for Jenkins plugins.	6.0.0-1
Git This plugin integrates Git with Jenkins.	4.10.3
Git client plugin Utility plugin for Git support in Jenkins	3.11.0
Git Pipeline for Blue Ocean BlueOcean Git SCM pipeline creator	1.25.3
GIT server Plugin Allows Jenkins to act as a Git server.	1.10
GitHub API Plugin This plugin provides GitHub API for other plugins.	1.301-378.v9807bd746da5
GitHub Branch Source Plugin Multibranch projects and organization folders from GitHub. Maintained by CloudBees, Inc.	2.11.4
GitHub Pipeline for Blue Ocean BlueOcean GitHub organization pipeline creator	1.25.3
GitHub plugin This plugin integrates GitHub to Jenkins.	1.34.3

Figure 9.Jenkins list of installed/ available plugins to enhance the tool functionality

Inclusion of all tools as plugins under a single window not only enhances the functionality of the tool but at the same time bridges the gaps between developers and operations team.
Next step includes the installation of Tomcat server to make war file of web application as depicted in following figure

(10)–

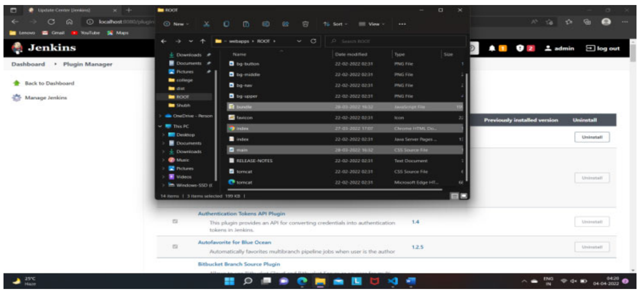

Jenkins
Dashboard
Plugin Manager
Back to Dashboard
Manage Jenkins
admin
log out
Previously installed version
Uninstall
Authentication Tokens API Plugin
Autofavorite for Blue Ocean
Bitbucket Branch Source Plugin

Figure10. Directory structure of Tomcat after inclusion of ToDoReact application code

Above figure (10) clearly depicts the directory structure of Tomcat to make a war file of the application. With the Jenkins Tomcat deployment plugin installed, it's time to create a new Jenkins build job that can build an application and deploy a package WAR file to Tomcat as displayed in the following figure (11) –

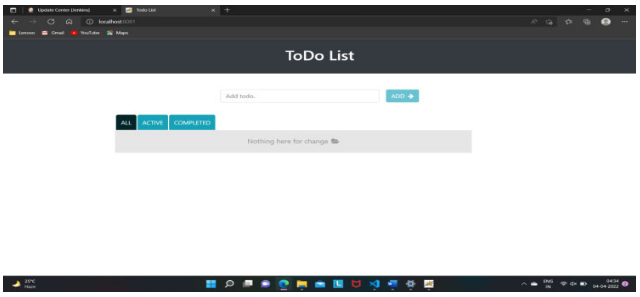


Figure11. Deployment of war files from Jenkins to Tomcat to run the application

Above figure (11) shows the successful implementation of a java web based application in Jenkins environment through different plugins. Any changes in the code requires the rebuild of application and restart of the Tomcat server as depicted in figure (12) below –

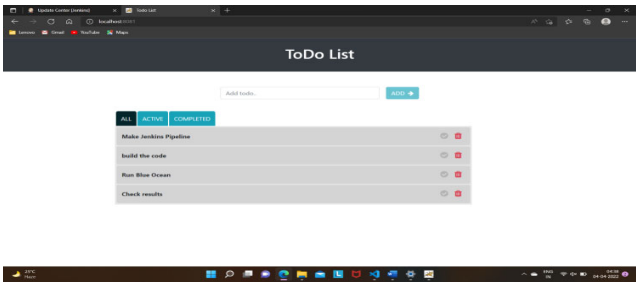


Figure12. Adding or deleting tasks and restarting of Tomcat server to include the changes

Above figure (12) shows the inclusion of different tasks after application deployment in Tomcat server under DevOps continuous environment.

7. Metric Selection and Evaluation to Analyze Software Quality

Software or system performance is always an essential criterion to decide quality of software. Software Quality describes desirable properties of not only software products but also its individual components as well as process followed. Different metrics are defined to quantify the quality of each software product and process. These metrics are required to evaluate the accuracy and performance levels of software. Performance measurement of developed software is also a significant step towards the improvement of software productivity and efficiency.

Software metrics are broadly classified into Process metrics (to assess different attributes of deployed methodology), Project metrics (to evaluate software project attributes like project status, cost, employees count or skill set etc.) and Product metrics (to assess or evaluate phase wise development of product) [14]. We have selected some already defined software metrics from existing literature for the purpose of software product and process validation. Considered metrics and their desirable outcome is shown in the following table –

Table 1. Software Metric Classifications for validation of the software developed.

Type of software metric	Software Metric	Expected Outcome or Results
Process	Risk Identification	High
Project	Project Defect Density	Low
	Release Deployment Frequency	High
Product	Process Productivity	High

Above Table (1) shows the inclusion or consideration of different software metrics along with their expected or desirable outcome.

The implementation or evaluation of DevOps development approach is performed through Jenkins automation tool along with inclusion of different required plugins. Much of the existing metrics generally considered evaluation of source code through different quantifiable measures. Our paper has taken specifically defined DevOps metrics [15] for the proper evaluation of underlying concept, process and their relationships. Table (1) above shows the metrics considered to assess or evaluate the performance of deployed software and its components. All these mentioned metrics are defined and evaluated below –

(i) System Risk Coverage Estimate (SRCE)

Many principles and methods were developed to conceptualize, assess and manage different software risks [16]. Software metrics provide a quantifiable vehicle for evaluating and managing quality factors along with early detection of risks involved in a given software product [17]. It will also be helpful in prioritizing risk in order to give them more weightage for consideration of removal. SRCE involves computation of risk coverage in individual software components followed by their sum up to get the same for the whole system or deployed software.

Risk coverage of individual components is given as –

$$RC = \sum_{i=1}^{n} W_i \qquad (1)$$

Where W is weightage assigned to the risk involved

And n is total number of requirements

The expression for system risk coverage estimate is given as –

$$SRCE = \sum_{i=1}^{n} RC_n \quad , \quad n > 0 \qquad (2)$$

Where n is total number of components/ modules in system

Estimate or measure of system risk coverage informs about percentage of risk coverage in designed test cases. Coverage of risk can be at higher with the consideration of more critical risks than testing the trivial ones.

(ii) System Defect Density Estimate (SDDE)

Defects or errors are the inevitable part of any software. Many of the defects can be ignored depending upon the severity of the defect / fault but developers should always look at incurred defects or defects that can be a risk to the software as inculcation of defects can hamper software performance drastically. In a study on defect density by US Authors [18], it was concluded that defect density has a greater impact on software quality and it also judges whether software is ready to deploy or not. Defects can be in many variants like post – deployment issues of code or any kind of error in connectivity to the back end databases etc.

Expression for defect density for individual software components is given as –

$$CDD = \frac{\text{Number of defects known}}{\text{Software size in KLOC}} \qquad (3)$$

Where KLOC (Kilo Lines of Code) = LOC /1000

To compute system defect density estimates for the software, we are to sum up individual components defect density to reach the final measure. It is given by the following expression –

$$SDDE = \sum_{i=1}^{n} CDDn, \; n>0 \qquad (4)$$

Where n is total number of components/ modules in system

Defect density, thus, affects the overall quality of the software.

(iii) System Deployment Frequency Estimate (SDFE)

Perera et al and Ziadoon Otaiwi et al in their work on DevOps Quality [19] [20], examined deployment frequency as one of the major goals of DevOps. In their study it was also noticed that higher deployment frequency leads to DevOps shine more than 40 times as compared to non DevOps performers. Deployment frequency, in general, refers to the frequency of code deployment with smaller size. As frequent deployments or releases with less requirement of changes is far better than less deployments with high requirement of changes. It is also directly correlated with continuous delivery and hence, measure of success for top-performing companies.

Expression for deployment frequency measure of individual components is given as –

$$DF = \frac{\text{Total number of Deployments}}{\text{Per unit Time}} \qquad (5)$$

Here Time unit is the function of project size. For example, if project size is Kilo Lines of Code (KLOC), deployment frequency can be weeks but in our research, size of data set is less so time unit is in hours.

Deployment Frequency Estimate for whole system is given by the expression below –

SDFE=$\sum_{i=1}^{n} DFn$, n>0 (6)

Where n is total number of components in the system

High value of SDFE indicates efficient and smooth functioning of the system.

(iv) System Productivity Estimate (SPE)

Continuous deployment is the practice of automatically deploying the software to the production environment. Increase in system productivity is an apparent benefit of continuous deployment [21] [22] Productivity is directly associated with process throughput. It indicates the amount of work done in a unit time interval. Thus it can be said that productivity is a clear measure of release count and success stories.

Productivity of individual components of software is given as –

P=$\frac{\text{Total number of User Stories Completed}}{\text{Time elapsed}}$ (7)

Where, User stories completed indicate the amount of work done. Time elapsed includes total time taken to complete the task.

Similarly, estimate or measure of productivity of the whole system/ software is expressed as below –

SPE=$\sum_{i=1}^{n} Pn$, n>0 (8)

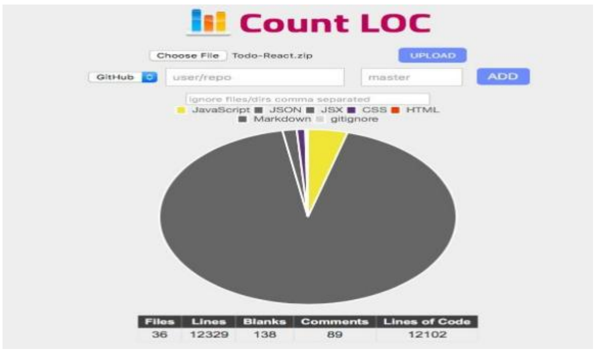
Where n is total number of components in the system.

To compute the quality of the produced or developed system, all above metric numbers should be high, except defect density. During this research work, defect density comes very low and that achieves the major requirement of DevOps quality deployment.

8. Java Based Web-based Application Todo-React As Data Set

A todo application, using React-Redux , is java based application that includes many features like user can add to do items in the list, added to do will be active by default and display under All and Active section. Also once to do is checked as completed it won't be displayed under the Active section. Again user cannot add a, to do task if that to do is still active (case-sensitive).

This Java web-based, ToDoReact application that keeps track of different status of tasks – active, pending and completed, has been taken as a sample project or case study to evaluate our selected set of DevOps metrics. Different parameters of code measurement is calculated as –



Files	Lines	Blanks	Comments	Lines of Code
36	12329	138	89	12102

Figure13. LOC, Components count for current data set with online Count LOC [23]

Above figure (13), shows calculation for the ToDoReact data set like total number of components, lines of code in total along with comments and blanks included. Following table covers different descriptive measures of selected project or application –

Table 2. Description of selected data set with different measures

Project Name	Size of Project (LOC)	Number of components
ToDoReact (Web-based Java Application)	12102	36

Table (2) above shows different descriptive measures of sample java based web applications taken or designed for the purpose of a data set to evaluate DevOps performance through a selective set of metrics.

9. Results and Discussions

Metric analysis of releases and final deployment is essential for measuring progression of project, expected outcomes and for overall success of the project. Experimental analysis of different features of DevOps is performed on the basis of metrics defined above and tabular results are shown for the underlying data set in table (2).

Risk coverage in software includes risk identification, assessment and management. Inclusion of risk was established as a scientific field around 40-45 years ago. Many advances have also been made to previously developed ways of risk assessment. [16] Risk coverage estimate for individual components and for the whole system measures total amount of risks covered in terms of risks tested, resolved, executed and skipped or not being executed. Risk coverage also covers the fact that some of the tests have more priority over others so have more weightage. Risk metric executes high weights risk first with the consideration of being more critical risks. Some risks are also not even executed or simply skipped on the basis of their weightage / priority. Table (3) below includes this detailed risks coverage analysis –

Table3. Individual Components and Risk coverage estimate for whole system with current data set

Total number of risk sets	**Number of Risks not tested**	**Count of broken risks**	**Number of risk sets skipped**	**Number of executed risk sets**	**Risk coverage %age**
1200	200	150	50	800	66.67%

Risk metric table (3) above shows detailed analysis of risks covered for the underlying data set of Todo project based on eq (1) and eq (2). DevOps hybrid model approach followed for this data set clearly shows high value of risk coverage percentage which indicates better inclination towards risk coverage. Risk assessment and proper coverage ensures increased customer satisfaction along with fast product delivery.

Another metric, defined to assess quality of process followed for product development, is defect density estimate. Defect density, alternatively, covers risks included in a project in the form of defects or bugs confirmed in the application or project. System defect density estimate or SDDE metric, measures total number of defects available divided by the size of project in terms of kilo lines of code. Number of total components in our data set is 36 and defects introduced in the system or individual components is 1,2 or none and considering 2 defects in each components on an average, gives total defects as 36*2=72, also, kilo lines of code are given as 12.102 KLOC. Thus, Component Defect density is calculated as using eq (3) –

$$CDD = \frac{2}{12.102} = 0.1652$$

For System estimate of defect density present in application or project, as given by eq (4) using CDD value as well –

SDDE = 0.1652 * 36 = 5.949

Table (4) below shows the values or figures of defect density computed for the current data set –

Table4. Component and system defect density estimate with the underlying data set

Project Name	**Total number of Components**	**Project Size (LOC)**	**Total no of available defects**	**System defect density estimate (SDDE)**
ToDoReact	36	12102	72	5.949

SDDE value as computed in the above table (4) indicates a good estimate of defect density by the DevOps tool chain hybrid model that yields a good quality product, on the other hand. Also verified from the above table, less value of defect density indicates measuring or defect detection much early in software development making the process more persistent and reliable.

Next project based metric, included in current research for the testing of software quality through usage of DevOps hybrid model approach, is deployment frequency. Development processes taking much time for deployment of intermediate or final products are less adaptable to customers as compared to frequent deployments or releases. Generic value of deployment frequency also allows major changes in the software without big hampering of budget or schedule. SDFE measures the total number of deployments per unit time. Using eq (5) & (6), deployment frequency for different components and for the whole system is calculated as –

$$SDFE = \frac{36}{3.6} = 10$$

System frequency of code deployment comes out to be 10 for total time taken for development is 3.6 hours. Following table (5) shows value of deployment frequency for the considered web based java project to verify the quality delivered with hybrid model approach –

Table5. System deployment frequency estimate for current data set of Java application

Total number of Components	Project Size (LOC)	Total number of deployments	Total time taken to deploy (hr)	System deployment frequency estimate (SDDE)
36	12102	36	3.6	10

As per the results obtained in table (5), it appears that DevOps hybrid model deploys 10 components or modules per hour. The high number of deployment frequency is achieved and this also indicates increased number of releases with more acceptability of major changes or updates.

Next metric for process improvement is system productivity or efficiency estimates. Productivity metric is used to measure or track the team efficiency of tasks done. These metrics are different from other quality metrics as these can also be used to get feedback from customers or employees for any project/ team. So, productivity metrics are concerned with system throughput which is given by the total number of user stories completed per unit time. User story, also referred to as epics, is an expressed requirement from the viewpoint of the end-user. This count for individual components on an average is 25, which gives the total number of user stories to be 25 multiplied by 36. Table (6) below computes system throughput or system productivity estimate (SPE), in other words –

Table6. Component and system productivity estimate (SPE) with the current data

Total number of Components	Project Size (LOC)	Total number of user stories	Total time taken to complete (in weeks)	System productivity estimate (SPE)
36	12102	900	13.5	66.7

In the above table (6), SPE, for current data, shows an increased value for SPE or throughput with DevOps automation tool approach. DevOps, thus, increased efficiency or productivity in terms of better throughput. Results shown in above tables (3)-(6), clearly show much better outputs in terms of good increased reliability, high risk coverage, better defect density along with much higher deployment frequency and system throughput or productivity.

10. Conclusion and Future Work

This research focuses on implementation of already proposed hybrid model of DevOps automation tools that is used to deliver quality software product with reduced deployment and delivery time. The present research work is the combination of Jenkins CI/CD tool and JDK web based application with Tomcat deployment server. This composite framework consists of inclusion of different required plugins in Jenkins. This inclusion of Continuous environment of DevOps not only reduces the development and delivery time but also accepts frequent changes in software and delivers continuously with the same quality and speed as shown in terms of tabular metric values. These continuous features of DevOps make it the latest buzzword of IT industry. Tabular results can be of great help to our young researchers/ students to understand the mode of operation of DevOps and its automation tools. Software developers will also benefit faster along with accurate selection of tool chain for speedy and quality delivery. Current research can also further be extended to cover more metric comparison for DevOps with other existing traditional models.

References

[1] Poonam and Pooja Mittal, DevOps Tools at different stages of Software Development: Analysis and Review, National Conference on Emerging Trends in Smart Computing, ETSC-2019, Organized by Department of Computer Science and applications, Maharshi Dayanand University, Rohtak (Hry), ISBN 978-93-80544-35-9

[2] Poonam Narang, Pooja Mittal, Hybrid Model for Software Development: an Integral Comparison of DevOps Automation Tools, Indonesian Journal of Electrical Engineering and Computer Science (IJEECE), IAES Publishers, ISSN 2502-4752, Scopus, SJR 2020 (Q3 0.241), Vol 27, No 1, July 2022, pp 456-465.

[3] Debois P., (2008), Agile infrastructure and operations: how infra-gile are you? Agile 2008 Conference, IEEE, Toronto, ON, Canada, ISBN: 978-0-7695-3321-6

[4] Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., Kuvaja, P., Mikkonen, T., Oivo, M., & Lassenius, C. (2019). DevOps in practice: A multiple case study of five companies. Information and Software Technology, 114, 217-230.

[5] Khan AA, Shameem M. Multicriteria decision-making taxonomy for DevOps challenging factors using analytical hierarchy process. J Softw-Evol Proc. 2020; 32(10):11-13, e2263.

[6] Leite L, Rocha C, Kon F, Milojicic D, Meirelles P. (2019), A survey of DevOps concepts and challenge, ACM Computing Surveys (CSUR). 2019; 52(6):1-35

[7] Bolscher R, Daneva M. (2019), Designing software architecture to support continuous delivery and DevOps: a systematic literature review, ICSOFT. 2019: 27-39.

[8] Ronny Olguin (2019), DevOps Challenges and Implications, University of Murcia, Spain, 2019.

[9] Ramdin Jabbari, Nauman Bin Ali, Binish Tanveer, Kai Petersen (2016), what is DevOps? A Systematic Mapping Study on Definitions and Practices, ACM Digital Library, published in Proc. of The Scientific Workshop Conference XP2016.

[10] Trihinas D, Tryfonos A, Dikaiakos MD, Pallis G (2018). DevOps as a service: pushing the boundaries of microservice adoption. IEEE Internet Comput;22(3):65-71

[11] Forsgren N, Tremblay MC, VanderMeer D, Humble J (2018). DORA platform: DevOps assessment and benchmarking. International Conference on Design Science Research in Information System and Technology, Springer, Cham. 2018:436-440.

[12] S. W. Hussaini (2014), Strengthening harmonization of development (dev) and operations (ops) silos in its environment through systems approach, In IEEE 17th International Conference on Intelligent Transportation Systems (ITSC), 2014.

[13] Prashant Agrawal, Neelam Rawat (2019), DevOps, A New Approach to Cloud Development and Testing, International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), IEEE publications 2019.

[14] Norman E. Fenton and Shari Lawrence Pfleeger (1997). Software Metrics: A Rigorous and Practical Approach. PWS Publishing Company 1997.

[15] Pooja Batra, Aman Jatain (2021), Hybrid Model for Evaluation of Quality Aware DevOps, International Journal of Applied Science and Engineering, Chaoyang University of Technology, ISSN: 1727-2394.

[16] Terje Aven (2016), Risk assessment and risk management: Review of recent advances on their foundation, European Journal of Operational Research, Elsevier, Vol 253, Issue 1, pp 1-13.

[17] Dr Issa Traore (2006), Software Architecture, Chapter 6, EOW 415.

[18] Cobra Rahmani and Deepak Khazanchi (2010), A Study on Defect Density of Open Source Software, 9th IEEE/ACIS International Conference on Computer and Information Science, IEEE/ ACIS ICIS, Yamagata, Japan, 18-20 Aug 2010.

[19] Pulasthi Perera, Roshali Silva, Indika Perera (2017), Improve Software Quality through Practicing DevOps, 2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ITCer): 013-018.

[20] Alok Mishra, Ziadoon Otaiwi (2020), DevOps and software quality: a systematic review, Computer Science Review, Elsevier, Vol 38, 100308.

[21] Leppänen M., Mäkinen S., Pagels M., Eloranta V.P., Itkonen J., Mä ntylä .M.V., Männistö T. (2015), The highways and country roads to continuous deployment IEEE Software, 32 (2), pp. 64-72

[22] Parnin C., Helms E., Atlee C., Boughton H., Ghattas M., Glover A., et al. The top 10 adages in continuous deployment, IEEE Software, 34 (3), pp. 86-95

[23] https://codetabs.com/count-loc/count-loc-online.html (accessed 2022)

Poonam Narang, Research Scholar, pursuing PhD from the Department of Computer Science and Applications, Maharishi Dayanand University, Rohtak, Haryana under the supervision of Respected Dr. Pooja Mittal (Research Guide and Second Author). Author's Qualification is M.Phil. (CS), MCA. She had attended many National and International Conferences including Springer and IEEE and also published many research papers. She can be contacted at email: poonam.mehta20@gmail.com.

Dr. Pooja Mittal obtained her Ph.D. degree from Maharshi Dayanand University. Her area of research and specialization include Data Mining, Data Warehousing, and Computer Science. She had published more than 50 research papers in renowned International and National Journals and attended more than 30 Conferences. Currently she is working as Assistant Professor in the Department of Computer Science & Applications, Maharishi Dayanand University, Rohtak (Haryana). She can be contacted at email: mpoojamdu@gmail.com.