Challenges and Solutions in Implementing Continuous Integration and Continuous Testing for Agile Quality Assurance

Amit Bhanushali

Quality Assurance Manager, West Virginia University, Independent Researcher, West Virginia, United States of America
Email: *akbhanushali[at]mail.wvu.edu*

Abstract: ***As software development undergoes a transformation, it necessitates a corresponding adaptation on the part of quality specialists. The fundamental feature of agile approach is quality, which is assessed by both developers and customers in order to enhance the system's overall quality. This approach has the potential to enhance the quality of output, but at the expense of less involvement from the quality assurance team. The Agile methodology places emphasis on expediting the software development process while also reducing costs. Additionally, within the framework of Agile Quality Assurance (QA) and software development, Continuous Integration (CI) and Continuous Testing are key methodologies. The programmers ensure that changes to the code are regularly applied, examined, and verified throughout the whole development cycle. Within the Agile Quality Assurance (QA) paradigm, continuous integration and continuous testing are important approaches that offer routine code integration, automated testing, and quick feedback. They play a key role in the swift and iterative delivery of software of the highest calibre in Agile development setups. This study's major goal is to address the Problems and Solutions of Putting Continuous Integration and Continuous Testing into Practise for Agile Quality Assurance. Agile techniques, agile software quality assurance, continuous integration practices of agile software development, difficulties, and solutions for these practises have all been covered in this review article. Additionally, the practises for continuous testing in agile quality assurance, difficulties, and solutions have been studied in the following part.***

Keywords: Agile, Software Testing, Quality Assurance, Continuous Integration, Continuous Testing, challenges, solution

1. Introduction

The quality factor may be a key element and a sign of the development and success of agile in the software development process or it may be a sign of the failure of software production. To attain a greater degree of quality, it is crucial to keep an eye on the success component. **[1].**

Today, many businesses are looking at various quality assurance procedures in an effort to find a solution that will allow them to overcome the difficulties associated with maintaining quality in agile environments **[2],** despite the fact that it was shown to be effective in the past, it is not suitable for the Agile setting any more. A review of what can be accomplished at each stage of the software development life cycle (SDLC) is one area where emphasis needs to be placed in order to maintain the required level of quality, for instance. More formal and thorough technical evaluations, such design reviews and so on, should be included in this review. This evaluation guarantees that defects and other issues are detected and fixed early on, improving the quality of the final output. But the official review that is being undertaken at the moment has taken the place of this formal review process.

The newest programming techniques can be thought of as evolutionary, iterative, and incremental. This includes techniques like the Enterprise Unified Process (EUP), Rapid Application Advancement (RAD), Extreme Programming (XP), and Rational Unified Process (RUP), for instance [3]. A lot of contemporary processes are also quick. Quality experts must adjust when the underlying nature of programming development changes. This study explains common agile software development techniques and shows how implementing them produces software that is considerably higher quality than what traditional software teams generally generate.

As Ampler reported, **[4],** The creation of test-driven may help agile approaches achieve the goal of producing high-quality software. According to Ampler, "quality is an inherent factor of agile," which seems reasonable," due to the significant amount of pressure that is placed on agile improvement teams to write test cases before creating code **[4].** Testing and quality go hand in hand since finding defects in a product before it is sent to customers is the primary objective of testing. This provides the opportunity for the product's developer to improve the product's quality by addressing any issues that were discovered **[5].** McBreen has highlighted quality as an agile value **[6]** as well as the flexibility to adapt to changes in development software. This suggests that a key challenge for agile quality assurance is to deliver tested, functional, and client-certified software at the conclusion of each new release.

The necessity for a wide variety of software products is growing along with the complexity of software on a daily basis. This necessitates the provision of a potent instrument that can balance output and quality. The practise of applying software metrics to the software development process and to a software product is a crucial task that necessitates study and discipline and that provides knowledge of the status of the software development process and/or product in relation to the goals to be achieved. This discipline is known as quality assurance, and it is the primary driver of success for every software engineering project. The quality assurance activities are what result in the qual because it enables the development of software with a minimal set of requirements and facilitates frequent changes in those needs, agile methodology is currently one of the main approaches employed by the majority of software industries. Even though the procedure might create the product rapidly, we cannot ensure its quality until we incorporate SQA activities into the process. [7]

McBreen (2003) [8] presents his perspective. The process of creating software that can adjust and respond to changes in accordance with the changing requirements of the client may be referred to as agile quality assurance. This shows that a key element of quality assurance in the agile methodology is the regular delivery of software that has been evaluated, verified as functioning, and approved by the client at the end of each iteration. The agile approach to quality assurance surpasses conventional software quality assurance methods by addressing quality concerns in a more advanced manner.

2. Theory of Ongoing Integration and Ongoing Testing

2.1 Permanent integration

The practise of continuous integration has previously received a lot of attention, and it is now well established and understood. Numerous more SCM-related operations need continual integration, and we anticipate seeing them develop in accordance with our expectations now that the foundation has been established. This will result in the creation of new bestpractises, some of which might have specific SCM-related sub practises to clarify the best practises. Asklund, Bendix, and Ekman (2004) [9] offer a first attempt to identify SCM sub practises appropriate for an agile setting in this paper. We also hope that these sub practises will evolve into more developed sub practises.

Kent Beck first introduced continuous integration as a software development technique in his book Extreme Programming Explained. This technique is frequently referred to by the acronym CI. **[10]** and on accelerating the speed at which developers may collaborate. The program is often modified by different developers, and each integration makes sure the product is functional by running automated tests and taking other precautions. The usage of continuous integration has been shown to improve software quality, testing effort and speed, software release frequency, and other factors, according to literature on software engineering and example cases.

A continuous integration (CI) server is often used to carry out continuous integration. When a developer publishes a change, CI servers immediately run a continuous integration build, show the results, and, if desired, inform the author of the results. This helps developers practice continuous integration. Although the majority of software projects that perform continuous integration do so using CI servers, the activity may also be carried out manually, as author in **[11]** represented.

The developer makes several changes to the source code, and these modifications often take place several times each week or on a daily basis. The code integration step is considered to be the most important part of the whole DevOps lifecycle. The process of continuously integrating new code into an old one involves the construction of new codes that enable the addition of new capabilities. Bugs are found quite early in the source code during this round of testing. Tools for unit testing, code review, integration testing, compilation, and packaging will be used by developers when writing new code for the programme. The functionality of the programme will increase thanks to this additional code. Continuous integration of this new code into the existing source code helps to portray any changes that end users may experience as a result of the upgraded code. Jenkins is a trusted DevOps tool that is frequently used for obtaining the most recent source code and transforming builds into executable forms. These transitions go place without a hitch, and the modified code is packed before being sent on to the next step, which is either the server used for production or the server used for testing. **[12]**

2.2 Continuous Testing

It is common practice for developers to do the continuous testing step before the continuous integration phase. Depending on the modifications made to the application code during the DevOps lifecycle, this phase may be shifted so that it follows the continuous integration phase. Now that the program has been constructed, it is being tested on an ongoing basis to find any issues. Docker containers are used in order to perform the simulation of a testing setting. Automated testing is a time and labor-saving tool for developers. The test evaluation process benefits from the reports generated by automated testing. It is now much simpler to analyse the test cases that were unsuccessful. The final test suite does not include any errors once the User Acceptance Testing (UAT) procedure has been finished. Utilizing these tools, one is able to schedule the running of test cases within a certain amount of time. In order for the source code to be updated, the tested code is ultimately delivered back to the continuous integration phase. This is carried out to ensure that the code runs without errors. The two crucial procedures that must be carried out to ensure that the application code will undergo continuous upgrades are continuous testing and integration. During the Continuous feedback phase, these improvements are evaluated. **[13]**

3. Agile Methodologies

From the beginning, when requirements are obtained, to the very end, when the software product is delivered, tested, and user input is gathered, the agile methodology follows the software development life cycle. The less documentation that is provided in the project and the greater emphasis on coding are what set agile techniques apart from other methodologies [14].

Scrum, extreme programming (XP), and other agile methodologies are founded on the application of tried-and-true techniques that are widely recognised to raise the calibre of software development. One may argue that employing best practises is done so as to make it easier to integrate software quality assurance (SQA) into the project. A key support structure for the project is provided by the quality assurance (QA) activities that take place during the software development process [15]. Since the 1990s, the agile methodology has received substantial discussion in a variety of academic publications, including books, essays, and journals. However, there hasn't been much research done on the subject of quality control in agile software development methodologies.

In a nutshell, it is a framework for developing software that is predicated on a high frequency and rapidity of iterations in order to both provide the program and accommodate modifications requested by the customer **[16].** Agile practices are becoming more popular in the modern day as a means of coping with the dynamics of corporate expansion. This is due to the fact that agile enables a rapid reaction to ever-evolving requirements and delivers a product of higher quality and greater speed **[17].** Because of their capacity to manage changing needs, emphasis on client and developer collaboration, and early software delivery, agile methodologies have gained traction in the commercial world since the late 1990s. The software development industry has mostly utilized agile approaches, which offer certain advantages in managing time and system user needs **[18].**

Numerous advantages of agile approaches include increased client happiness, increased project success, and higher development quality. Agile development approaches improve coordination between teams. These techniques promote quick market entry; however, the delivery process is slowed by a lack of coordination between developers and the operating team. Agile companies carry out development, testing, and deployments independently. These tasks take a lot of time, which caused the release process to be delayed **[19].** The operation and testing teams are not given a significant amount of attention in agile techniques. The processes of testing, delivery, and deployment are each the responsibility of these teams. The product is developed by the development teams at a significantly quicker speed, which causes other teams to lag behind, which might cause the delivery process to be delayed. As a result, problems and errors manifest themselves early in the process of producing a product.

The DevOps methodology enhances communication, delivery, performance, and integration between operational staff members, testers, and developers.**[20]** The objective of DevOps is to enhance customer satisfaction by means of enhanced quality and uninterrupted delivery. The use of DevOps by several European banks resulted in a noteworthy 25 percent enhancement in their provision of online services **[21].** The implementation of DevOps relies on the use of an automated deployment pipeline, hence reducing the repetitive manual tasks associated with continuous integration **[22].** Development and operations are simply two parts of the DevOps process **[23].** However, other stakeholders, including developers, testers, analysts, as well as personnel responsible for database management and security, actively participate in the implementation of DevOps practices **[24].**

DevOps can assist in overcoming the difficulty of continuous delivery. Digital behemoths like Amazon and Netflix already employ DevOps to deliver precisely crafted, customer-focused software solutions to the market [25]. The application of DevOps concepts to implement continuous delivery eliminates traditional testing, monitoring, and code integration, hastening the release of the user's product. By utilising reusable items and promoting the widespread use of software management systems, the DevOps paradigm aids in continuous delivery [26]. Continuous delivery principles are the foundation of DevOps practises. Fast release reduces the time it takes to develop an application and deliver high-quality software **[27].** Testing on a more general level is performed in software companies to assess the aim and establish the quality of the program. On the other hand, a quick quality check should not be provided for continuous integration. Testing is done manually and there is no automation of any of the test cases, hence the rate at which errors are found will gradually increase. Because of the amount of work and time that is required for this procedure, the delivery of the goods will be delayed. Continuous testing is one strategy that may be used to help address these difficulties. Continuous testing enables the delivery of timely feedback with no involvement from humans. Automated testing is a component of continuous testing. This monitoring and improvement of test case quality is accomplished via automation **[28].** Continuous testing strategies are used in testing activities that are a part of a DevOps strategy. This helps to identify flaws and errors within a variety of software components. Testing is approached in a manner that is distinct from that of traditional testing thanks to the DevOps principles, which may be thought of as a set. Testing was one of the stages of the software development life cycle process prior to the emergence of DevOps. However, testing is not carried out during the project's whole phase, which lasts from beginning to completion. With DevOps, testing should be implemented from the start to guarantee that the software is of a high quality and to share responsibility for that quality between the development team and the operations team. **[29].**

3.1 Process Improvement for Agile Software

A thorough study of agile processes leads to the conclusion that agile approaches are a collection of procedures and activities that reduce the time needed to create software programmes and offer cutting-edge techniques for accommodating swiftly altering business requirements. These relatively new QA methodologies should eventually grow into established software engineering standards. Enabling Reusability in Agile Software Development, however, has shown **[30]** 2.8) Agile Practises: 2.9) is what makes it possible to complete the software development process more quickly.

3.2 The Framework for Evaluating Agile Methodologies

All agile approaches have remarkably similar processes across their numerous iterations since they are predicated on the same set of four agile ideals and 12 agile principles. It is interesting to note that even the developers of agile methodologies now accept the use of techniques from other agile methodologies as long as they are suitable for the particular situation at hand [31]. In reality, Kent Beck discusses the shortcomings of extremism in both the first version of his book on extreme programming (XP) and in his XP masterclasses. An in-depth examination of agile methodologies reveals that these methods employ a number of different models based on real-world scenarios to address the same issues. When thinking about the process of software development, lean development (LD) adopts a manufacturing and product development metaphor, as shown by the approach for evaluating everything that is discussed in this article. Through the lens of control engineering, the Scrum framework tackles the software development process.

Extreme programming is a paradigm that views the software development process as a group activity where developers collaborate. The methodology of adaptive systems development, also referred to as ASD, is used to approach software development projects from the perspective of complex self-adaptive systems [32]. A small number of the available agile methodologies were used to show the assessment process. There is a lot of subjectivity involved in the choice of methodological elements. The goal of this work is not to provide an exhaustive taxonomy of approaches. for a more thorough taxonomy. As a result, the objects included here were chosen to show the similarities among different agile methodologies. By revealing these similarities, developers who are unsure on which agile technique to choose will be better equipped. While employing the right technique in your software development project may not instantly result in project success or the production of a high-quality product, doing so can lead to project failure. Thus, there is in knowing.

4. Agile Software Quality Assurance

Quality was described by Ambler (2005) [4] as being a direct result of the agile methodologies themselves from the perspective of agile development. Due to these characteristics of agile quality, comprehensive documentation is no longer necessary; yet, this results in the word agile quality becoming somewhat vague and becoming more challenging to define [33]. Since the bulk of agile methods only offer a small number of suggestions on how various quality attributes should be checked and maintained, integrating testing approaches with agile operations can be difficult. This is true even though the most common SQA activity nowadays is assessing numerous quality attributes. According to Itkonen et al. (2005, p. 202) [34], the main challenges that agile principles present for testing from a traditional point of view were highlighted. They emphasised the necessity to conduct testing in brief cycles of time while avoiding exceeding the permitted testing duration as one of the issues that must be solved in order to satisfy the criteria of the agile concept of continuous delivery of useful software. The most challenging part of "responding to change even late in the development" is that testing cannot be done on realised needs. This poses a problem. Due to the agile method's reliance on face-to-face interaction, it is feasible for engineers and business participants in the testing process to experience communication breakdowns that could result in disagreements. Additionally, they asserted that the most crucial sign of success is having software that works, therefore information on quality is necessary both early in the development process and throughout. Itkonen et al. (2005) concluded that the "simplicity-for-sake" approach makes the testing technique easily debunked.

Itkonen et al. (2005) observed conflicts in agile testing while taking such concepts into account when it comes to the basic testing principles. One of the core characteristics of testing, for instance, is independence. However, with agile approaches, developers create the tests for their apps, and testers either swap roles with developers or become developers themselves. It is a sign that testers are not sufficiently independent because they are integrated into the development team. Additionally, they should refrain from testing their own code because it is challenging to find errors and testing does not determine whether the final product satisfies or comprehends the needs of the customer. Additionally, testing requires individuals that have a specific set of skills and knowledge in order to be successful and effective. On the other side, consumer testing is viable to attain quality if they have unique experiences to handle or conduct testing, which is regrettably not always the case. Customers might test products, nevertheless, to ensure quality. Finding the correct test result and locating programme non-conformances are processes referred to as the "oracle problem" This presents still another difficulty. Agile approaches use a large percentage of automated tests, which raises the question of whether these tests are sufficient to find defects in the code. The destructive approach, which focuses on investigating and identifying problems, is one of the most well-known methods of software testing. According to [35], conformity testing is carried out to make sure that an entity complies with a specific requirement and/or legal requirement. This study's goal is to assess a software system's effectiveness in fulfilling the required specifications and to gauge how faulty it is [36]. Agile techniques may lead to system faults despite passing unit tests because they place a higher priority on the description of product qualities than on the investigation of fundamental causes. However, the testing procedure not only looks for flaws but also collects data that could be utilised to improve the product's quality. Although it does not reveal the level of product quality reached or make it easier to evaluate the attained quality, the agile method heavily relies on following established protocols and checking their adherence.

4.1 Agile Development Processes are Produced by Testing Practices

It is possible to argue that the implementation of extra testing techniques ultimately results in the adoption of agile development procedures. In the following chapter, the work of Itkonen et al. (2005) will be examined in order to provide a more thorough explanation and clarity regarding these shortcomings and challenges. Based on the "heartbeat, iteration, and release time horizons" in their research report, Itkonen et al. (2005) used a temporal pacing model to identify quality assurance techniques in current agile approaches. The Cycles of Control (CoC) structure, which was established by Rautianen (2004) **[37],** in the context of an agile development approach.

4.2 Control Framework Cycle

The CoC framework is a general example of a frame that may be used to portray incremental and iterative development of software. Because of this, the CoC framework can be used to illustrate agile development. The design is based on the idea of time pacing, in which a predetermined period of time is divided into several time periods of varying lengths, each of which has a budget and a deadline. **[38].** A control point is located at the conclusion of each individual segment and is used to conduct an analysis of the process. Alterations to the plans may also be made if it is determined that they are required. Since adjustments can only be made at specific control points, persistence and the creation of flexibility are both made possible, allowing for simultaneous plan changes and responses to the environment's changing conditions at predetermined time intervals.The flow of product development is determined by these intervals, also referred to as time frames. A time box's schedule (due date) is determined in accordance with the idea of time pacing, but its scope (creating functionality) is not. **[39].** In the event that it is not feasible to meet all of the product criteria by the end date that has been set, the scope of the project will be reduced in order to ensure that it will be ready by the end date that has been set. As a result, "Therefore, demands need to be ranked in order for the team to be able to make scope suiting on their own." An illustration of the cycles of control building pieces may be seen in Figure 1.

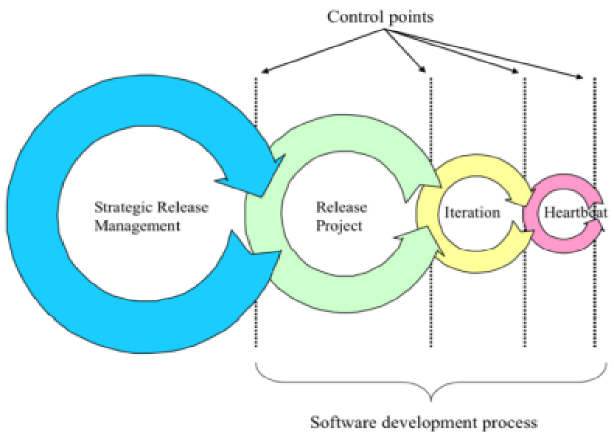

Figure 1: Cycles of Control building blocks [37]

"The creation of long-term strategies for a company's product and project portfolios is included in strategic release management. It acts as a vital link between the actual process of product creation and strategic decision-making [39]. Each product release is managed successfully inside the release project cycle as a "time-boxed project" that is carried out. Each project is organised into time-boxed iterations, wherein a portion of the eventual product release is created [39]. Heartbeats serve as a timer and a regulator for daily activities [37]. Figure 2 illustrates this by showing the cycles as a timeline. This shows how the strategic release management time horizon spans two distinct release projects, incorporates three distinct interactions, and synchronises the work with daily heartbeats.

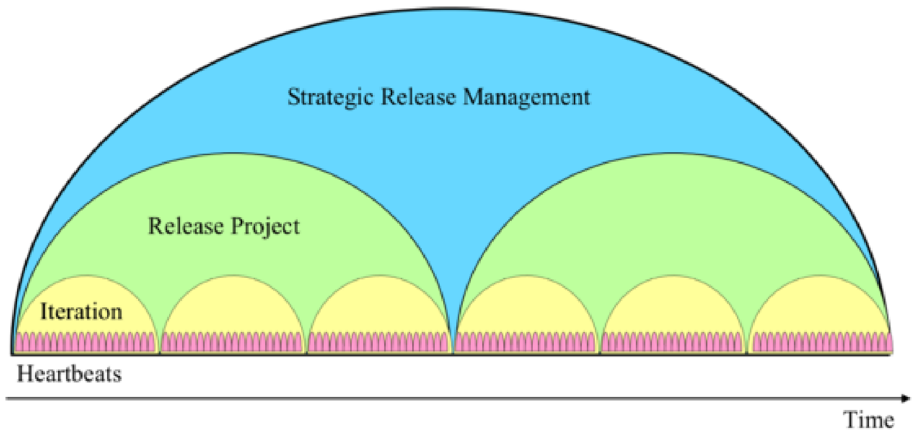

Figure 2: Timeline of Cycles of Control [37]

a) Quality Assurance Heartbeat

No additional testing phase is used to postpone the Heartbeat QA operations. Instead, these tasks are completed as part of the design and coding responsibilities during the implementation phase. This is true regardless of whether a developer or a tester is in charge of carrying out these activities. These activities provide the developers with immediate feedback, allowing them to lead the development process in the right directions. Heartbeat quality assurance practises include methods that "put quality into a piece of functionality during its implementation," which means that the implementation chores are not deemed complete until these QA procedures have been completed. These procedures are incorporated into the design and coding duties [34]. They serve as a solid foundation for the agile development process and make it their objective to guarantee that each and every development task is effectively accomplished because these practises give the developers immediate feedback. Automated unit testing acts as the benchmark and the heartbeat of quality assurance. Each and every piece of code that is written by the developers must be subjected to unit tests, and advancement is only given consideration once the tests have been completed and passed. "Rhythm is the key," and "heartbeat activities are managed and monitored according to the heartbeat rhythm." The secret is rhythm [34]. Each heartbeat must also be successful for the many functions to interact with one another and coordinate their actions. When it comes to agile approaches on the pulse time horizon, QA is a highly potent discipline. The "heartbeat time horizon" provides the foundation for the majority of quality assurance strategies, as indicated in Table 1. This is due to the emphasis on unit testing, frequent builds, code and design inspections, and quick integration cycles in the methodologies that have been presented.

Table 1: QA procedures for agile approaches on time horizons [34]

	"Extreme Programming"	"Feature Driven Development"	"Crystal Clear"	"DSDM"
Release Iteration	Evaluating the acceptance test results	"Separate system testing"	-	• "Integration, system and acceptance testing inside each timebox" • "User testing" • "Evolutionary prototyping" • Reviews of documents
Heartbeat	• "Test-driven development" • "Continuous integration" • "Pair-programming" • "Acceptance testing" • "Collective code ownership" • "Coding standards" • "Simple design and continuous refactoring" • "On-site customer"	• "Unit testing" – • "Regular builds" – • "Design inspection" – • "Code inspection" – • "Individual code ownership"	• "Automated tests and frequent integration" • "Side-by-side programming" • "Osmotic communication" • "Easy access to expert users"	• Unit testing • Reversible changes • Active user involvement

On the iteration time horizon, quality assurance covers procedures that are focused on accomplishing the iteration's goals. This encompasses all implementation, testing, and review phases that don't apply to every single cardiac rhythm feature. Iteration time horizons are responsible for a large number of jobs that are performed by professional testers. These duties include evaluating the system's dependability, performance, and other quality characteristics. Functional testing is just one kind of testing that may be performed on a system. In general, the activities that professional testers engage in are ones that are only tangentially related to the development process. These experts are responsible for writing and carrying out testing for the whole of the iteration, and they must coordinate their efforts with the developers. It is essential to keep track of the work's growth, progress, and continuous communication of high-quality information since iteration tasks are time-boxed. One way to do this would be via heartbeat meetings. As indicated in Table 1, there are much fewer quality assurance practises on the iteration time horizon of agile methodologies than there are on the heartbeat time horizon. At this level, there are not many processes or activities that are specified specifically for the purpose of quality assurance. For instance, Extreme Programming (XP) is reliant on robust pulse methods, and the iteration time horizon is solely responsible for measuring progress **[40, 41, 42].**

"Ensuring the product's quality on the time horizon of the release is the primary objective of the release quality assurance process." **[34].** Examples of tasks include testing that cannot be finished within the allotted time for an iteration, assignments given to a different testing group, and testing in various environments. A good general strategy for engaging quality assurance releases is to have distinct stabilisation iterations. Since the stabilisationiteration assesses the quality of the work produced in earlier iterations at the end of the release project, this stabilisation phase is not considered to be iteration quality assurance. Additionally, it implies that some quality problems are not discovered until the process' very last iteration.

According to Itkonen et al. (2005) [34], Table 1 shows that it is challenging to locate any quality assurance processes in agile approaches on the release time horizon. The dynamic systems development method (DSDM) states that there may be times outside of iterations when separate/split acceptance testing processes are necessary, for as on the precise release time horizon, in huge projects, or because of contractual limits [43]. Even in DSDM, these circumstances are thought to be unusual.

b) Quality Assurance Iteration

Quality assurance is concerned with tasks and activities that are not performed for each implemented feature individually at a heartbeat rhythm on the iteration time horizon. Instead, with a goal-focused approach, these actions are controlled and tracked over an iterational time horizon. This includes all practises for implementation, testing, and review necessary to ensure that the iteration's end product is of sufficient calibre. Iteration tasks are time-boxed since each iteration has a predetermined length. Throughout the iteration, the testers must prioritise their tasks. As a result, it's crucial to keep an eye on the task's development and consistently share reliable information, for instance through heartbeat meetings. Without current knowledge, it is challenging to decide on the scope of the iteration QA tasks to complete in order to achieve the desired product quality by the end of the iteration. There are far fewer QA procedures in agile methodologies on the iteration time horizon than there are on the heartbeat time horizon. Table 1 demonstrates that there are only a few well-established methods for assuring and evaluating the quality of the software increment produced during each iteration. Less defined than those on the heartbeat time horizon are the procedures on the iteration time horizon. Some methodologies, such as XP, almost entirely rely on sound heartbeat practises and only view progress monitoring as important throughout the iteration time horizon [41]. The other methodologies, which have less strict heartbeat practises, acknowledge the necessity of using system testing to evaluate the quality that has been attained over the course of iteration, but they do not give any specific instructions on how to go about doing so. For instance, the FDD's single recommendation on how to accomplish this goal is to decide which builds and how frequently to submit to independent system testing. **[44].**

c) Quality Assurance Release

Release quality assurance is used to ensure that a product's quality will be upheld over the duration of the release time horizon. This includes reviewing test findings and any other high-quality data gathered from the various iterations, as well as ways to direct the development project based on the knowledge acquired (for instance, planning upcoming iterations). On the release time horizon are the responsibilities of a distinct testing group, testing in numerous settings, and testing that cannot be finished within the iteration timetable. These are some of the types of testing. Having a separate stabilization iteration at the conclusion of the release project is a typical practice that is used to include release quality assurance. Because the stabilization iteration analyses the quality of the work done in the iterations that came before it, this is not the quality assurance for the iteration. Because of this, some quality hazards are not uncovered until the very final iteration of the process. It is quite difficult to identify any quality assurance processes in agile methodologies on the release time horizon. For instance, we were unable to locate any quality assurance procedures among the sample techniques shown in Table 1 that would be appropriate for the release time horizon. In some cases, independent (acceptance) testing procedures outside of iterations, i.e., outside the release time horizon, may be necessary, according to DSDM. These circumstances may arise as a result of contractual limitations or in the case of extremely large projects **[43].** However, even according to the DSDM, such instances are regarded as uncommon.

4.3 Agile Testing Improvements for QA

Agile techniques are dependent on the involvement of users, but they do not include a significant amount of experience in destructive testing. Only user acceptance tests are used for testing in some approaches, like XP, which offers a set of effective development activities with the aim of producing outstanding software. Some approaches, on the other hand, do not provide such a list of tasks and, as a result, acknowledge the need for specific testing methods not just at the integration level but also at the system level and acceptance level. This leads to the conclusion that heartbeat quality assurance procedures have the potential to be enhanced, for example, by adding the role of an independent tester who tests each completed feature alongside the developer [34].

Agile testing is well-represented by the session-based exploratory testing (ET) example on the iteration time horizon. "Exploratory Testing (ET) is an informal approach to test design where the tester actively shapes test designs while running tests," according to Veenendaal (2018) [45]. The tester uses the knowledge gathered from the testing process to continuously improve and upgrade test cases. [45]. It is a novel approach built on experiences where test design, learning, and execution are carried out concurrently, and the outputs from these processes are immediately applied to the process of building more tests. As a result, it does not depend on test cases that have been predesigned. Testers use their knowledge in two different ways, according to Itkonen et al. (2012) [46]: for test design and as a test oracle for spotting failures. This indicates how well testers' knowledge works during exploratory testing sessions. The use of session-based exploratory testing enables the efficient management of testing within concise time frames, making it suitable for brief iterations. Furthermore, as previously stated, the essence of exploratory testing facilitates the cultivation of the requisite negative mindset essential for effective testing. In some scenarios, there is a need for testing during the release time frame. However, it is often more advantageous to include several quality assurance (QA) methods throughout the iteration time frame and the heartbeat phase. This approach ensures the availability of quality information at an earlier stage and mitigates the risks associated with poor quality during the first stages of development.

Agile methodologies place significant emphasis on customer or user participation and tend to exclude a number of damaging testing procedures. With the exception of user acceptance tests, which fall under the client's purview, certain approaches, like XP, offer a thorough set of development practises that prioritise the production of software of a satisfactory quality without relying on significant testing. It should be highlighted, however, that other agile approaches lack a thorough set of standards and do not take into account the requirement for specialised testing practises at the integration, system, and acceptance test levels. The Dynamic Systems Development Method (DSDM), a well-known illustration, requires the implementation of testing at several levels throughout each iteration. Based on the existing body of literature, it is evident that there is a lack of comprehensive advice about the effective integration of destructive and independent testing procedures into agile development processes. Our study demonstrated that using time horizons to describe the development process and the procedures used facilitates the identification of areas where testing processes may be improved. Gaining comprehension of the temporal scope of heartbeat and its associated methodologies provides the necessary alignment for the actions of developers and testers. There is potential for improvement in the quality assurance (QA) methods related to heartbeat. One such enhancement is the incorporation of an independent tester who collaborates with the developer to test each completed feature. By relying on independent destructive tests rather than solely the developer's constructive methods, this feature provides instant feedback on the quality achieved. Session-based exploratory testing is an example of agile testing that takes place within an iteration time horizon [47]. Exploratory testing is a testing technique that appears to embody the agile ethos because it doesn't rely on specified test cases. It is best to use session-based exploratory testing in conjunction with brief iterations since it enables testing to be managed in condensed time limits. With the exploratory method, you can assess the entire system or, for instance, the interactions between several distinct pieces. Our initial research demonstrates the value of exploratory testing for testing apps from the end-user's perspective and quickly discovering critical flaws. The exploratory nature of testing makes it easier to develop the disruptive mentality required for effective testing. Exploratory testing may also be used to persuade employees of businesses or others with deep subject knowledge to participate in testing. In some circumstances, testing tasks may be necessary on the release time horizon. Agile methodologies for the release time frame have not yet been found. In many cases, it may be beneficial to incorporate as many QA procedures as possible on the iteration and pulse time frames in order to offer quality information early and help avoid quality risks.

5. Agile Software Development Practices with Continual Integration

5.1 Continuous and Software integration

Software integration is the process of linking various software subsystems or components to form a coherent and cohesive system **[48].** The inclusion of a software development lifecycle is an important component within the realm of software engineering, since the development process often encompasses many stages and involves collaboration among a group of engineers. Software integration is often performed as a distinct stage inside the conventional software development lifecycle, occurring subsequent to the completion of software implementation.One of the twelve guiding principles of Extreme Programming (XP), Continuous Integration (CI) is a software development methodology that was initially introduced in 1997 **[49, 50]** which is,

- 40-hours week
- Coding standards
- Collective ownership
- Continuous integration
- Metaphor
- On-site customer
- Pair programming
- Refactoring
- Simple design
- Small release
- Testing
- The planning games

It simply indicates that every developer combines their work on a regular basis—at least once every day. This approach guarantees that minor components are incorporated as soon as they are finished and ready to be a part of the system, preventing the emergence of complexity [51]. It is important to automatically create and add test cases for the entire system, including the recently introduced components, in order to execute the Continuous Integration strategy. Additionally, the programme must be built and tested automatically, and the developer must receive quick feedback on new codes before adding them [52,53,54]. Any comments at this time will be considered by the developer as soon as they are received. When a programmer is still getting to know his or her programmes, this helps in the process of identifying flaws and issues. To reap the rewards of adopting CI, developers must modify their typical daily practises of software development. People frequently need to refrain from contributing code, fix problematic builds immediately away, create automated tests, make sure that all written tests and inspections pass, do private builds, and avoid getting code that is broken. **[55].**

5.2 The Role of Continuous Integration in Software Quality and Testing Lifecycle

Code modifications made by different developers are often merged once per day while using continuous integration since it is so frequent. Because the program is produced and at least unit tested on every integration, it is possible to guarantee a fundamentally adequate level of quality. Also, since developers constantly obtain the most recent modifications from other developers, it is more probable that they would encounter difficulties with those changes, which can then be eliminated faster in comparison to when the changes would only be accessible for other developers after a period of days or weeks **[10].**
Continuous integration is one method that may be used to enhance manual software testing, particularly the duration of the testing cycle. When there is not continuous integration, testers have to wait for the program to be produced before they can test it, then they have to deploy the product into their testing environment, run the software, and finally get to the section of the software that needs to be tested. All of these phases have the potential to fail, forcing testers to wait for developers to address the problems, which in turn increases the amount of time required before software modifications can be tested and approved. Testers simply need to deploy the most current software build, and they may have a greater level of confidence that the program will work up to the point when it reaches a section that has to be tested since continuous integration ensures that the software is always in a generally functional condition. If the program is also automatically delivered to the testing environment as part of a continuous integration build, then the manual deployment phase may also be skipped by the testers. Continual integration builds are a kind of build that is performed continuously. Testers are able to concentrate their efforts on the sorts of tests that are most effectively carried out manually by humans, such as non-functional testing and exploratory tests. If it is known that the program is functioning properly as a whole as a result of a continuous integration build that includes an extensive set of tests, then the software may be tested **[10].**In addition to testing for non-functional requirements like capacity and security, testers can also confirm the program's usability. Comprehensive automated testing combined with manual testing of the programmeas a whole has the potential to produce better software quality than just performing manual tests.

5.3 Continuous integration techniques used by developers

The basic steps in continuous integration will be dissected in this section, along with each step's individual components. The company is now considered to be adopting continuous integration in the software development process as a result of putting these concepts into practise.

- **Committing code frequently:** The focal point of the continuous integration process. When it comes to committing their work to the common code repository, developers shouldn't wait more than one day before doing so. The following are some of the options available to developers for simplifying this process: 1- Instead of making sweeping modifications to a number of components all at once, make just little adjustments. 2- Make a commitment after finishing each individual activity, supposing that the chores can be completed in a matter of a few hours apiece.
- **Don't commit broken code -** Programming that causes errors of any kind when it is incorporated into a continuous integration build is referred to as "broken code" [56]. Developers should first build and test their code locally on their own PCs before making changes to the common code repository. They should wait until all tests and inspections have passed before committing any code.
- **Fix faulty builds right away:**A problem with deployment, the database, or compilation could lead to a flawed build. Anything that prevents the construct from proclaiming success is acceptable [57]. The project's top priority should be fixing a flawed build, and the relevant developer should respond to such concerns very away.
- **Write developer tests that are automated:** Tests must be automated in order for them to run effectively in a CI environment. It should cover the entire source code as well.
- **All automated tests and inspections must pass in order for a build to be successful**: The most important CI requirement for software quality is this. The integration build includes coverage tools that are used to help identify source code that lacks a corresponding test case. To run automated inspections and check general design and coding standards, additional tools are also employed.
- **Run private builds**: Thanks to CI technologies, programmers can keep a local copy of their workstations' local copy of the software from the shared code repository. In order to make sure it doesn't fail, they can use a recent integration build locally first before merging it to the main integration build server.
- **Avoid receiving broken code by using CI technologies to help you report cade failures**: One of the most important features of CI is its capacity to give developers quick feedback. The condition of the code is continuously updated, and if it is broken, no developer should check it out from the shared code repository. The person in charge should start developing a fix as soon as feedback reveals that the code is flawed. If not, the immediate input offered by CI is lost. **[54]**

5.4 CI Factors Influencing Test Cycle Time & Software Quality

The degree to which continuous integration is implemented varies widely amongst software projects. A continuous integration build may involve just a few processes, such as an automated software build and automated unit tests, but it can also include many extra phases, such as automated acceptance testing, automated deployment, or software metric computation. Other variables, such as the time it takes to complete a continuous integration build or the frequency with which software changes are integrated, may also have an influence on a continuous integration implementation and its impact on software quality and test cycle time. The following measures are intended to provide excellent internal and external software quality as well as a quick test cycle time. The frequency with which software updates are integrated is the first critical element. When modifications are incorporated more often, flaws in those changes are discovered and remedied sooner. Furthermore, when integrating more regularly, the size of changes is lower, which decreases the risk of each integration and aids in the effective detection of problems. Developers are more inclined to integrate their modifications less often if the length of the continuous integration build is long **[57].** As a result, it is also crucial that continuous integration builds complete quickly. For providing high software quality and quick test cycle times, the execution of automated tests as a component of a continuous integration build and their volume are also crucial. Software testers may concentrate on specialized testing tasks like exploratory testing instead of repetitive testing activities like regression testing by employing automated tests **[10].** Continuously computing software metrics, often known as automated inspection, helps improve the internal quality of software. Metrics like code coverage or the quantity of duplicate code may reveal problems with the source code's quality and make that quality obvious **[57].** It is also simpler to take corrective action if the origins of the code's issues have already been identified via automated examination. Automated software deployment into testing environments may help ensure excellent software quality by offering a tried-and-true method of program deployment. The likelihood that the software deployment will be carried out differently each time or for various environments if it is done manually is quite high. The risk of deployment into a production environment is decreased by offering an automated deployment mechanism and testing it by running it as part of a continuous integration build.

5.5 Quality elements in the development process

Which quality characteristics are impacted by the continuous integration of software was determined using the ISO standard definition of characteristics. [58] categorised quality criteria according to how they related to various stages of the software development lifecycle. Since continuous integration is a well-known development methodology, this study largely focuses on the quality features that fall under its umbrella. A thorough list of traits and their accompanying measurements are shown in the following table:

Table 2: Quality elements in the development process and their measures

Attribute	Measure
Time to develop	Time to complete a feature's implementation and get it ready for testing
Introduced bugs	The total number of bugs found in either the newly written code or the impacted older portions of the code.
Time to deliver	Time to complete the process of implementing and testing a feature, and then make it available for usage by customers.
Test quality	The tests need to cover all of the features and find the majority (if not all) of the issues.
Documentation	It is necessary for the document to include all of the work that was done by the developers.
Change management	Alterations to the specifications have to be acceptable and shouldn't demand an excessive amount of time and labor.
Cost model	It is calculated by comparing the costs of preparing the new setup to the amount of money that will be saved as a result of putting the new adjustments into effect.

The following table presents a comprehensive overview of the many quality aspects associated with software frameworks, highlighting the distinctions between conventional integration approaches and continuous integration methodologies.

Table 3: Quality before to and after using continuous integration

Comparison criteria	Before continuous integration	After continuous integration
Time to develop	The requirements for the whole release are worked on by the developers.	The work of the developers is focused on a particular need.
Introduced bugs	When testing is done only after completing a substantial portion of the code, the number of problems is much higher than when testing is done after merely implementing tiny portions of the code.	It is not necessary to wait until the whole release is ready in order for each feature to be tested and corrected when it is finished being developed.
Time to deliver	After completing the release and putting it through its paces	After completing a single prerequisite and checking if it's been met,
Test quality	• All of the releases as a whole are put through testing. • The environment used for testing is not the same as the one used in production.	Once a feature has been incorporated, testing is performed on its separate components, and the results of the testing are sent back to the developers as soon as they are available. The testing environment and the production environment are almost indistinguishable from one another.
Documentation	• The testing process encompasses each • Every one of the releases in their entirety. • The environment that is used during testing is not identical to the one that is utilized during production.	• There is documentation of the needs, • The quantity of documents produced by the developers is low. • The automated tools create statistics and data on the developed features based on the requirements, the testing results, and the generated defects, etc.
Change management	• Change is only acceptable after going through a lengthy procedure and receiving permissions • Modifications are implemented by means of a whole new patch or release.	Any change may be implemented at any moment, but the owner must first communicate newly outlined needs to the business analysis and software development teams.
Cost model	The amount of time and effort that would be required to manually perform the software development and integration	The expense involved in acquiring a server and tools for continuous integration

6. Challenges and Solution of Continuous Integration Practices of Agile Software Development

In the Agile methodology, testers assume distinct duties that vary from those in older methods. In this context, the tester actively engages with all stakeholders. In a distributed context, the management of test cases becomes a challenge when the number of sprints rises, leading to a proportional growth in the size of the test suite. An examination of the existing scholarly works **[59, 60, 61, 62]** evident that the aforementioned concerns need attention with regards to testing practices within an Agile setting.

6.1 Testing activities in an agile testing lifecycle

Challenge - The issue at hand pertains to the ability of an Agile tester to engage in other tasks concurrently with their testing responsibilities. The available literature lacks a comprehensive depiction of the sequence of activities, including regression testing, within the Agile testing life cycle **[63, 64].**

Solution: A suggested Agile testing life cycle involves active engagement between the tester and other relevant stakeholders. The tester position is defined by its focus on conducting testing operations aimed at ensuring the delivery of a high-quality product to the client. Moreover, the significant impact of regression testing has been recognized.

6.2 In a distributed context, use agile testing

Challenge - The technique of pair programming is used in the implementation of agile testing in order to get a high-quality result. The matter at hand pertains to the methodology of conducting testing in situations when team members or customers are not physically situated in the same place. Another concern is to the process of forming pairs for the purposes of pair programming or testing **[65].**

Solution: A conceptual structure for a distributed environment **[66]** has been put out that includes answers for problems experienced by customers and dispersed team members. Particularly, refactoring **[67]** has been advised for managing dispersed obstacles by adhering to basic design principles. Furthermore, the issue of related difficulties in dispersed environments may be solved using a pattern that has an evolved solution from several best current solutions. Additionally, a self-centric strategy (common team member traits) has been suggested for pair programming and pair testing to help team members create a pair.

6.3 Testing for regression in an Agile setting

Challenge - Agile often involves adapting to change, therefore this might have a significant impact on the dependent modules **[68, 69]** and the quantity of test cases rises as a result. Regression testing is therefore more necessary. Therefore, in order to manage the larger number of test cases, a regression testing approach is needed.

Solution: A methodology for selecting regression tests **[70]** a model that is based on the user narrative graph and that includes links between user stories has been presented. In addition, two metrics known as average path value and average path length have been used in order to discover an ideal way out of all the paths that are included in the user narrative graph, including paths that do not already exist and paths that do currently exist. In addition to this, a tool for the suggested regression test selection approach has been developed and made available in Microsoft Excel.

6.4 Prioritization of test cases in a distributed environment

Challenge - The challenge is in managing test cases for a sprint in a distributed environment when there is a frequent need to react to change and when current user stories are impacted as a result of that change **[67].**

Solution: A linguistic strategy **[71]** which is based on the number of pauses detected in user stories for narrative priority and the number of nouns and verbs identified in user stories for test case prioritization, has been suggested for test case prioritization. In addition, a risk-based approach that is based on the identification of high hazardous stories has been suggested as a method for the prioritizing of test cases. Additionally, a tool for the suggested test case prioritizing approach has been developed and implemented in Microsoft Excel.

7. Continuous Testing Practices for Agile Quality Assurance

Saff and Ernst [72] introduced and popularised the idea of Continuous Testing (CT) as a way to reduce the amount of time that is spent when running tests. Additionally, Gamma and Beck were responsible for the idea. (2003) Gamma and Beck They listed running all of a project's tests automatically each time the project was built (a feature known as "auto-testing") as one of the advantages of a plugin. Running tests frequently is one of TDD's goals. However, the developer frequently has to leave his work in order to physically carry out the tests. You can continue to develop the codebase using contemporary IDEs like Eclipse or Visual Studio. Continuous compilation (also known as automated compilation or automated build) is a common term for this. By enabling the IDE to perform the build in the background while the developer is writing and/or saving the file, this method eliminates the waste of manually compiling the code after writing some source code. By doing tests as the developer is working, CT advances this approach. Running the tests doesn't require stopping what you're doing. The developer obtains prompt feedback as the tests run automatically in the background. Saff estimates that the amount of waste removed to be between 92 and 98%, and that this has a significant effect on the efficiency of programming job completion [74]. In [75], a practical approach to CT is shown. The practise of continuous testing (CT) is made easier by the broad variety of plug-ins that are readily accessible on the market and adapt to various integrated development environments (IDEs) and other tools. The majority of the tools are provided as extensions for modern integrated development environments (IDEs). The earliest tools were made especially for the Java programming language and the Eclipse Integrated Development Environment (IDE). For the platforms of Visual Studio and.NET, comparable utilities were developed. Tools are additionally available for programming languages like Ruby. Several organisations and individuals have created different Continuous Testing (CT) tools. These tools include Contester, an Eclipse plug-in developed by students of the Software Engineering Society at Wroclaw University of Technology, JUnit Max, an Eclipse CT plug-in, NCrunch, a commercial Visual Studio plug-in, Autotest, a Ruby continuous testing tool, Continuous Testing for VS, a commercial Visual Studio plug-in, AutoTest.NET, and Mighty Moose, a packaged version of AutoTest. The creators of integrated development environments (IDEs) today are more and more aware of CT's importance. Microsoft Visual Studio 2012, the most recent version, includes a continuous testing functionality. Only the two most expensive editions of Visual Studio 2012—Premium and Ultimate—allow access to this feature. As a result, the industry is aware of how important the CT practise is. Our goal is to build on this understanding and take use of any potential synergy that can arise from the combination of TDD and CT ideas in the context of agile software development. Additionally, we want to acquire empirical evidence to support the usefulness of applying this suggested practise in industrial settings.

In the field of software development, testing is incredibly important. There are major costs associated with the software testing process. To build and run the tests, the testing process necessitates a sizable time and effort investment. The cost of development is typically halved while engaging in testing activities. In order to address these issues, the use of continuous testing might be employed. The concept of continuous testing was first developed by Saff and Ernst in the year 2002. An experiment was undertaken to demonstrate that the use of continuous testing may lead to a reduction in the overall development time by around 15 percent. The experiment conducted demonstrates that the use of continuous testing may effectively reduce waste in the development process, particularly in terms of minimizing waiting time. Continuous testing involves the use of automation techniques and the strategic prioritizing of the testing process. Continuous testing is a more efficient and effective approach for the identification and detection of faults, requiring a reduced amount of time and effort. The author **[76]** define continuous testing as a quick approach for automatically and without human involvement detecting faults in test cases. It continuously runs tests to make sure the code is of a good calibre. These tests run automatically as programmers create new code. Continuous testing may offer continuous feedback by running tests in the background without involving engineers. Extreme programming and continuous compilation may be seen as extensions of continuous testing [77]. Continuous compilation provides quick feedback regarding compilation errors. Continuous testing, according to Siegel [78], is a practise in which a team of developers conducts tests manually and often throughout the testing process. Continuous testing is the practise of conducting tests frequently and early in the development process. Continuous delivery is dependent on continuous testing in order to improve product quality and ensure that it is mistake free.

8. Challenges and Solutions for Continuous Testing in Agile Quality Assurance

8.1 Challenges

Testing on a more general level is performed in software companies to assess the aim and establish the quality of the program. On the other hand, a quick quality check should not be provided for continuous integration. Testing is done manually and there is no automation of any of the test cases, hence the rate at which errors are found will gradually increase. The process will take longer than expected, which will create a delay in the delivery of the items. Continuous testing is one strategy that may be used to help address these difficulties. Continuous testing enables the delivery of timely feedback with no involvement from humans. Automated testing is a component of continuous testing. This monitoring and improvement of test case quality is accomplished via automation. Continuous testing strategies are used in testing activities that are a part of a DevOps strategy. This helps to identify flaws and errors within a variety of software components. Testing is approached in a manner that is distinct from that of traditional testing due to the fact that DevOps may be thought of as a set of principles **[79].**Prior to DevOps, testing was one of the stages of the software development life cycle process, but it is not performed for the entirety of the project. But with DevOps, testing should be present from the beginning and ensures the product's quality through shared responsibilities between development and operations teams.

Many studieslike **[80]** highlighted the obstacles that an IT company confronts while implementing DevOps methods. **Claps et al. [81]** The organization's CD adoption difficulties have been identified as plugins and CI. They also observed that the requisite skills and expertise for implementing DevOps principles might provide a barrier to enterprises. The researcher **[82]** Inappropriate architecture, manual testing, and aversion to change were found to be barriers to adopting continuous delivery. Because to tool constraints, continuous distribution has been suspended. Current technologies pose security risks during deployment and provide inadequate feedback during testing.

Adoption of DevOps approaches such as continuous delivery necessitates changes in company culture, position, and release-related duties **[19].**Many businesses find it challenging to create procedures that work, thus interviews should be conducted to determine the benefits and issues. System architecture, testing, and integration tools were found to be the key barriers to the adoption of continuous delivery under DevOps principles by Laukkanen et al. [83] after conducting an SLR. They argue that companies' strategies must change when they make the switch to DevOps and CD. According to a subsequent study by Laukkanen et al. [84], switching to continuous delivery may be challenging given complex design. Ullah et al. **[85]** executed a semi-structured method and identified a continuous delivery pipeline's security risk. Due to hacking and data theft from the CD pipeline, they draw attention to the internet danger. They suggested leveraging container technologies like Docker to solve this problem. Software testing and speedy delivery were the subjects of a case study and a semi-systematic literature review by Mäntylä et al. [86]. They asserted that hurdles to CD and testing include low test coverage performance, time restraints, and customer satisfaction. The author **[87]** a study was done with developers, and the results showed that the majority of developers are unaware of the significant business-related dangers that are posed by continuous delivery pipelines to development. The continuous delivery process is made more difficult by the large number of infrastructures and external dependencies involved.

Roche [79]explains the value of open channels of communication and close working relationships between the development and operations teams in the context of software quality assurance. He discovered that DevOps provides the finest solutions for testing and delivery. When it comes to testing, DevOps principles and features employ a number of methods that are different from those used in regular testing. It is guaranteed that there will not be any delivery delays by using metrics and prioritizing the test cases. The issue of adopting DevOps may be mitigated with the assistance of testing teams in a business. In order to facilitate automated deployments, releases, and monitoring, continuous testing infrastructures may be divided into testing groups. This makes it easier to quickly get feedback on the software's quality. **[88].**

Testing is not only conducted continuously during the agile development process, but it is also an integral part of the DevOps workflow. It provides the teams with an accurate and transparent outcome of the test **[89].** As per the Angara et al. **[90],** Testing efforts for DevOps should include doing more thorough research in scientific and academic literature. Testing may be seen in a variety of different ways thanks to DevOps **[91].**DevOps can help the development and operations teams become closely bonded. The author has explored a number of test technique DevOps organisation approaches, such as feature toggles, infrastructure testing, paring, destructive testing, etc. According to her, the business development and operations teams may decide to use these strategies depending on their needs. Testing in production under DevOps gives the development team ongoing input or feedback. The author's three fundamental methods are A/B testing, monitoring as testing, and beta testing for production testing. Benefits of DevOps include real-time monitoring of the production environment and management of anomalies as soon as they appear. Testing tools employed in a DevOps methodology help in the development of effective test generation methodologies, as shown by Yuan et al. [92].

DevOps professionals have trouble locating precise tools for continuous practise tasks. They also lose time and energy trying to select the appropriate testing tool [93]. Many businesses employ DevOps processes for testing services such unit testing, development-driven testing, and behavioural testing. The testing pipeline receives code right away for further quality improvement. Testing should be a part of DevOps from the very start of the development process. Testing towards the project's conclusion has an effect on its quality. Furthermore, discontinuing continuous delivery without project testing is an option.

Mohammad, Sikender Mohsienuddin [94] Achieved greater performance during DevOps continuous testing by choosing the appropriate tools and technology. The tools save time and effort by automatically detecting issues during testing. A case study on the implementation of DevOps at a New Zealand company was conducted by Mali Senapathi et al. [95]. The authors contend that improving tester performance requires learning new tools and technologies. They found that collaboration between the development and testing teams increases effectiveness. Additionally, the adoption of DevOps closes knowledge gaps and enhances code quality. As part of a multi-perspective research methodology, Wiedemann [96] conducted a case study, interviews, and a workshop with IT specialists. For effective DevOps teams, they highlighted a number of competencies. Automation was given top priority while testing. Knowledge of automation is essential because manually writing and running test cases takes longer to identify and fix the root of an issue.

The author described how testing teams might overcome challenges in a DevOps transformation. The development team may provide rapid feedback if they establish a structure for continuous testing. The infrastructure for continuous testing and cooperation between development and operation enable a quick release and increased test coverage. Blogs cover topics like test environment management and performance testing as a code. On measures, however, that help teams improve their testing work, there isn't a lot of information available [97]. All services, from development to release, are impacted by or covered by DevOps testing. DevOps promotes security and performance testing. The author suggests that monitoring may be helpful in testing to get immediate feedback on technical services. To solve the issues of agile development, DevOps testing was created. DevOps testing refines quality and improves communication among all stakeholders. In academic or scientific papers, including case studies, there isn't much systematic research on DevOps testing.

Architecture that is strong and dependable enables testability and deployability. It facilitates getting quick feedback from the operation and development teams. Despite this, there is little study on how DevOp affects software architecture. DevOps methodologies work to quickly implement change in production in order to achieve high quality [98]. These techniques get rid of the structural obstacles to transformation. According to many software professionals, monolithic and other architectural styles/types are inapplicable to DevOps and CD [99]. DevOps, according to their argument [100], is not a suitable fit for highly connected design.

In order to pinpoint a number of concerns, the researcher [101] spoke with software vendors. They found that obsolete or legacy architecture can have an impact on the adoption of DevOps. Data collection from various technologies is difficult because of integration problems in these antiquated infrastructures. A case study on the IT group was conducted by Silva et al. [102]. They claimed that reference design makes it easier for employees to work together and solve business-related issues in a DevOps environment. The amount of operational effort was lowered by 50%, they discovered, thanks to the development and operations teams' interaction. When new modules are added to the production pipeline, DevOps reduces failures by 3%. According to the author, teams are driven to develop new apps by less errors rather than by focusing on defect rectification. To determine the approaches that may be employed for continuous architecture and DevOps, Taibi et al. [103] conducted a rigorous mapping analysis. They claim that the testing and deployment cycle is slowed considerably in microservice systems by continuous code reworking.

Due to its capacity to facilitate quick delivery and offer useful feedback, a sizable number of businesses are adopting the DevOps strategy. On the subject of DevOps transformation, numerous research studies have been conducted. Although this study focuses on continuous testing and how architecture affects the move to DevOps, it does not go beyond these two areas. In respect to Continuous Testing (CT) and Continuous Delivery (CD), there is a dearth of research that precisely studies the impact of architecture on production settings. There are few thorough studies that give an in-depth understanding of continuous testing and delivery within the context of the DevOps ecosystem. The researcher **[104]** Please provide a comprehensive compilation of the advantages and difficulties associated with the implementation of DevOps, as well as the specific obstacles encountered by DevOps teams in their pursuit of continuous testing. Nevertheless, the aforementioned study failed to address the resolution of difficulties pertaining to the adoption of continuous delivery, as well as the influence of architecture on DevOps settings. Further study is required to examine the influence of major corporations on these subject matters.

8.2 Solution

This study aims to identify the potential procedures or methods that might be used to address the issues associated with continuous testing and DevOps.

Table 3: Challenges' Solution

Solution	References
Participation of the customer in confirming the objective and requirement	[29]
Gathering of relevant needs	[97]
Using particular/specific tools	[105] [106] [107]
Use only non-commercial tools	[108]
Early and ongoing testing	[97]
Sprints are constantly being developed.	[109]
Communication and collaboration	[94] [96] [106]
Automation	[90]
Processes and protocols	[110] [111]
Understanding of responsibilities and duties	[112]
Cross-functional environment	[113]
Proper test case execution and clear results	[114]
There is no fear of failure or change.	[115]
Program for team exchange to share knowledge	[97]
Seminars, training, and workshop	[116]
Jenkins, Selenium, and version control tools are used.	[108]
Full-stack development and decision-making abilities are required.	[107]
Management and group members' assistance	[109]
Infrastructure development	[108]
Verification tests Execution	[90]
Utilization of cloud services	[108]

In **[29],** the author focuses on testing practises in a DevOps environment. Testing guarantees the quality of the services and software in DevOps. Customers increasingly demand quick responses and excellent apps. Additionally, automation is the most crucial stage to guarantee constant feedback. Teams strive to automate as much as they can under DevOps principles. Automated testing provides quick feedback by pointing software flaws. For instance, testing is required for each incremental release of a product under the new agile manner of working. To minimize labor and human error, automate as many tests as you can, including unit, integration, and acceptance tests. Continuous testing and monitoring were necessary for the creation of sprints and their release to customers. Additionally, continuous testing checks and confirms that the program is bug-free and that the objective is meeting client requirements.

In **[105],** the authors noted that a communication issue arises as a result of the geographic separation between the testing and development groups. They indicated that a skill-exchange program would enhance knowledge of testing and DevOps. Adopting DevOps involves using specific testing tools and improved communication channels. In [**115**], The inventor of DevOps emphasised the necessity of testing teams having a cross-functional ecosystem. When testers and developers work together, the effectiveness of unit and UI tests may be improved. Jenkins, for instance, automates code testing using a continuous integration server to save a considerable amount of time and work. [113] The authors claimed that their research could aid IT professionals in understanding the concept of DevOps. They said that cooperation between testing and development teams, as well as automation, are essential for applying DevOps concepts.

In **[109]**, The researcher noted that less resistance to change and stable management structures may facilitate the adoption of DevOps. The development operations team had previously operated independently and with different management structures. Contrarily, DevOps demands that each of these stakeholders strengthen the management structure. They found that DevOps and continuous testing face a serious challenge in maintaining outdated infrastructure. Because maintaining legacy systems is time- and money-consuming, organisations should adopt new tools and technologies. The author enhances the test automation procedure for continuous testing and development activities at [112], a Finnish software company. They discovered that the testing cycle can be shortened by utilising the most recent tools and services. In [97], the authors identified Norwegian DevOps companies' testers as the source of the issues. They contrasted DevOps characteristics with the frequency of tester challenges. They found that resources for collaboration, monitoring, and testing needed to be improved. Collaboration improves test coverage, which raises the quality of DevOps. In cross-functional teams, accountability encourages excellence. Developers and testers may switch roles and share technical information to bridge the knowledge gap and improve understanding of the testing process.

8.3 Other challenges and solution [117]

There are several challenges to overcome while trying to incorporate continuous testing in DevOps, including the following:

8.3.1 Management of Test data

Challenges - The management and upkeep of pertinent test data that effectively simulates real-world situations may be difficult.

Solution - The data may be made more relevant and reflective of real-world situations by using synthetic test data.

Challenges - It might be difficult to create and manage test data that accounts for a variety of conditions.

Solution -Techniques such as data masking, data anonymization, and the development of synthetic data may be used to provide test data that is both realistic and consistent with privacy regulations. In this respect, technologies such as Delphix and other TDM (Test Data Management) tools might be of use.

8.3.2 Management of Test environment

Challenges - Particularly difficult is the task, when dealing with applications of a complex nature, of ensuring that the testing environment is trustworthy and consistent.

Solution - Utilizing containerization technologies such as Docker is one way to help guarantee a dependable and consistent testing environment.

Challenges -It may be difficult and time-consuming to set up and maintain test environments that are accurate representations of production.

Solution - Automation of environment provisioning and configuration may be achieved via the use of infrastructure as code (IaC) and containerization technologies (such as Docker). Kubernetes and other tools may assist in the management of containerized environments.

8.3.3 Maintenance of Test automation

Challenges - Scripts for test automation may easily become complicated and difficult to maintain, particularly if the application continues to be updated.

Solution - Keeping test automation scripts up-to-date and relevant requires doing routine maintenance and reworking on a consistent basis

Challenges -As the size of the application increases, it may become more challenging to maintain and scale the automated test suites.

Solution - The use of modular test design and the utilization of frameworks such as Selenium, Appium, or TestNG may enhance the maintainability and scalability of test automation scripts. Additionally, it is advisable to take into consideration cloud-based testing solutions as a means to enhance scalability.

8.3.4 Integration Testing

Challenges -The complexity of testing relationships across different components and systems might provide challenges in Agile contexts.

Solution -For the validation of API and microservices interactions, contract testing and consumer-driven contract (CDC) testing are advised. Tools such as Pact and Spring Cloud Contract have the potential to assist in the achievement of desired outcomes.

9. Conclusion

Even while some of the agile practices have been around for a while, the agile methodologies themselves are relatively new and have gained a lot of traction in the business world. There is a huge knowledge gap among developers on the calibre of the software being produced. Developers must be knowledgeable about how their agile approaches may be updated or adapted in order to provide the highest quality work feasible. Continuous integration has been shown to significantly enhance software quality as a whole, prompting software manufacturers to adapt their development techniques in that direction. A large number of the hazards related to the process of integrating software were decreased as a result of integrating parts of the developed software continuously as soon as they are accessible.

The goal of this study was to investigate the level of QA coverage offered by the leans for agile software development that Itkonen et al. (2005) presented. This group of authors looked at the parallels and discrepancies between the agile and plan-driven methodologies' technical approaches. The theoretical foundations of software quality have been developed with a focus on the ISO 9126 quality perspective and the fundamentals of quality assurance, with the conventional viewpoint of quality being used as a starting point in order to move closer to this goal. Then, by contrasting the agile concepts with their corresponding challenges and by contrasting the fundamental testing principles with inconsistent practises in ASWD, it has been possible to identify the challenges and weaknesses of ASWD in connection to SQA. To ascertain whether ASWD complies with SQA, both of these comparisons were made.

For instance, it has been noted that agile methodologies don't apply explicit quality measures when it comes to their guiding principles. This lack of direct quality metrics was acknowledged as a potential weakness in the method. Additionally, it was determined that certain skills, the oracle problem, a destructive approach, and the independence of testing were in opposition to how agile approaches are now used. By including extra testing techniques that are not initially addressed in the agile methodologies itself, ASWD operations could be improved. This is possible by relying on the difficulties and drawbacks mentioned in the previous line. For the iteration time horizon, an independent tester position has been proposed, and for the heartbeat time horizon, session-based exploratory testing is being examined. It has been determined, after great consideration, that testing techniques must be covered during the course of these two times. This article lists the following challenges that come with agile testing. In order to determine whether or not improvement is actually essential, taking into account the differences between the single methods that more or less involve explicit testing activities in their agenda, it is first necessary to provide evidence of the sufficiency of existing SQA practisesutilised for existing ASWD. This is done to assess if improvement is genuinely required or not. Second, more investigation is needed to ascertain whether testing practises are efficient in agile development while also being able to meet the quality requirements that are imposed by traditional ways of doing business. For instance, two potential approaches to consider are behaviour driven development (BDD) and acceptance test driven development (ATDD).

In this day and age of DevOps, continuous testing is very essential to assuring product quality. It makes it possible to deliver software more quickly, it lowers the likelihood of problems occurring during production, and it improves the program's overall quality. Continuous testing in DevOps entails selecting the appropriate test automation tools, developing an efficient test automation framework, integrating testing into the DevOps pipeline, and adhering to DevOps best practices. These are the steps that must be taken in order to implement continuous testing.

10. Future Scope

In this particular piece of study, the continuous integration and testing methods of agile software development were the only topic of attention. In order to go on with this study in the future, further agile methods such as pair programming and combining the impacted quality criteria into a single quality framework may be investigated as potential options.

References

[1] Javed Iqbal, Mazni Omar, Azman Yasin, "Empirical Study of Agile Methodologies and Quality Management Success Factors in Pakistani Software Companies", 9th Knowledge Management International Conference (KMICe) At: Miri SarwakMalaysia , July 2018

[2] Rafaela Mantovani Fontana, Victor Meyer, Sheila Reinehr, and Andreia Malucelli. 2015. Progressive Outcomes. J. Syst. Softw. 102, C (April 2015), 88-108

[3] Kruchten, P. 2004. The rational unified process, 3rd edition. Reading, Mass.: Addison-Wesley Longman, Inc

[4] Ambler, S., (2005), Quality in an Agile World, Software Quality Professional, Vol. 7, No. 4, pp. 34-40

[5] Parvez Mahmood Khan , M.M. Sufyan," BegMeasuring Cost of Quality (CoQ) on SDLC Projects is Indispensible for Effective Software Quality Assurance", International Journal of Soft Computing And Software Engineering (JSCSE) e-ISSN: 2251- 7545,2012.

[6] McBreen, P., Quality Assurance and Testing in Agile Projects, McBreen Consulting, [online]. Available from: http://www.mcbreen.ab.ca/talks/CAMUG.pdf [Accessed 2017].

[7] Almustapha Abdullahi Wakili (2020). QUALITY ASSURANCE PRACTICES IN AGILE METHODOLOGY. International Journal of advance research in science and engineering, Vol no. 9, Issue no. 10

[8] McBreen, P. (2003). Quality assurance and testing in agile projects. McBreen Consulting. Retrieved January 12, 2006, from http://www.mcbreen. ab.ca/talks/CAMUG.pdf

[9] Ekman, T., & Asklund, U. (2004). Refactoring aware versioning in eclipse. Electronic Notes in Theoretical Computer Science, 107, 57-69.

[10] Humble, J., & Farley, D. (2011). Continuous Delivery. Reliable Software Releases Through Build, Test, and Deployment Automation. Addison-Wesley

[11] Shore, J. (2006). James Shore: Continuous Integration on a Dollar a Day. Retrieved April 17, 2019, from jamesshore.com: https://www.jamesshore.com/Blog/ContinuousIntegration-on-a-Dollar-a-Day.html

[12] Poornalinga, K. S., & Rajkumar, P. (2016). Survey on Continuous Integration, Deployment and Delivery in Agile and DevOps Practices. International Journal of Computer Sciences and Engineering, 4(4), 213-216.

[13] Lai, S. T., & Leu, F. Y. (2016, July). A Version Control-based Continuous Testing Frame for Improving the IID Process Efficiency and Quality. In 2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS) (pp. 464-469). IEEE

[14] Sharma, S., Sarkar, D., & Gupta, D. (2012). Agile processes and methodologies: A conceptual study. International journal on computer science and Engineering, 4(5), 892.

[15] Sagheer Maria, Zafar Tehreem, Sirshar Mehreen. (2015). A Framework For Software Quality Assurance Using Agile Methodology. INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 4, ISSUE 02,

[16] Iqra Zafar, Aiman Nazir, Muhammed Abbas, "The Impact of Agile Methodology (DSDM) on Software Project Management". International Conference on Engineering, Computing & Information Technology (ICECIT 2017 At: Kuala Lumpur, Malaysia, March 2018

[17] B. Boehm and R. Turner, "Using risk to balance agile and plan driven methods," Computer, vol. 36, pp. 57-66, 2003

[18] Deki Satria, Dana Sensuse, HandrieNoprisson, "A Systematic Literature Review of the Improved Agile Software Development", 2017 International Conference on Information Technology Systems and Innovation (ICITSI), September 2017.

[19] Gokul Patil, Nisha Magar, Vaishnavi Gangurde, Zarka Khan, Srushti Magar. "Enhancing software automation using DevOps", International Journal of Research in Advent Technology, Special Issue, ICATESM 2019

[20] G. Hackett, "Survey research methods.," Personnel Guidance Journal, vol. 59, no. 9, 1981

[21] Gregory, J., and Crispin, L., 2015", More Agile Testing: Learning Journeys for the Whole Team, Addison-Wesley, Upper Saddle River, N.J

[22] Ali, Nauman Bin, Kai Petersen, and Mika V. Mäntylä. - Testing highly complex system of systems: an industrial case study. Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, 2012

[23] A. Dyck, R. Penners, and H. Lichter - Towards definitions for release engineering and devops. In Release Engineering (RELENG), 2015 IEEE/ACM 3rd International Workshop on, pages 3–3, May 2015

[24] S. K. Bang, S. Chung, Y. Choh, and M. Dupuis. - - A grounded theory analysis of modern web applications: Knowledge, skills, and abilities for devops In Proceedings of the 2Nd Annual Conference on Research in Information Technology, RIIT '13, pages 61,62, New York, NY, USA, 2013. ACM

[25] Zhu L. Bass L. and Champlin-Schar;G: "DevOpsandItsPractices: (EEESoftware;Vol :33No:3 ; pp:32 34 :)" 2016

[26] J. Wettinger, V. Andrikopoulos, and F. Leymann, "Enabling DevOps Collaboration and Continuous Delivery Using Diverse Application Environments,‖ Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9416, pp. 107–116, 2015

[27] Toh, M. Zulfahmi, Shamsul Sahibuddin, and Mohd Naz'riMahrin. Adoption Issues in DevOps from the Perspective of Continuous Delivery Pipeline." Proceedings of the 2019 8th International Conference on Software and Computer Applications. 2019.

[28] B. Fitzgerald and K.-J. Stol. Continuous software engineering and beyond: Trends and challenges. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014, pages 1–9, New York, NY, USA, 2014. ACM

[29] Faber, Frank. Testing in DevOps." The Future of Software Quality Assurance. Springer, Cham, 2020. 27-38.

[30] S. Sukhpal and C. Inderveer. (2012). Enabling Reusability in Agile Software Development.

[31] Beck, K. et al. (2004). Manifesto for agile software development. Retrieved from agilemanifesto: http://agilemanifesto.org/

[32] E. Mnkandla, .B. Dwolatzky. (n.d.). Defining Agile Software Quality Assurance

[33] Mnkandla, E. &Dwolatzky, B. (2007). Agile software methods: State-of-the-art. In I. Stamelos& P. Sfetsos (Eds.), Agile software development quality assurance (pp. 1-23). Hershey, PA: Information Science Reference

[34] Itkonen, J., Rautiainen, K. &Lassenius, C. (2005). Towards understanding quality assurance in agile software development. In H.E. Andersin, E. Niemi & V. Hirvonen (Ed.), ICAM 2005. Proceedings of the International Conference on Agility (pp. 201-207). Helsinki, Finnland

[35] Zhou, X., Govindaraju, K., & Jones, G. (2019). Fractional nonconformance based conformity testing. Computers & Industrial Engineering, 135, 402-411

[36] Hasan, S. M., Islam, M. S., Ashaduzzaman, M., & Rahaman, M. A. (2019, December). Automated Software Testing Cases Generation Framework to Ensure the Efficiency of the Gesture Recognition Systems. In 2019 22nd International Conference on Computer and Information Technology (ICCIT) (pp. 1-6). IEEE

[37] Rautiainen, K. (2004). Cycles of Control: A temporal pacing framework for software product development management (Licentiate thesis). Helsinki University of Technology, Helsinki, Finland

[38] Eisenhardt, K.M. & Brown, S. L. (1998). Time pacing: Competing in markets that won't stand still. Harvard Business Review, 76(2), 59-69.

[39] Rautiainen, K. &Lassenius, C. (eds) (2004). Pacing software product development: a framework and practical implementation guidelines. Helsinki University of Technology Software Business and Engineering Institute Technical Reports 3

[40] Beck, K. 1999. Embracing change with extreme programming. Computer, 32(10), 70-77.

[41] Beck, K. 2000. Extreme programming explained (2nd ed.). Stoughton, MA: Addison-Wesley

[42] Jeffries, R., Anderson, A & Hendrickson, C. (2001). Extreme programming installed. Boston, MA: Addison-Wesley

[43] Stapleton, J. (1997). Dynamic systems development method. Harlow, England: Addison-Wesley

[44] Palmer, S.R., J.M. Felsing 2002. A Practical Guide to FeatureDriven Development. Upper Saddle River: Prentice-Hall.

[45] Veenendaal, E. (2018). Test techniques for the test analyst [e-book]. Retrieved from http://www.erikvanveenendaal.nl/site/wp-content/uploads/Test-Techniques-for-the-Test-AnalysteBook.pdf

[46] Itkonen, J., Mäntylä, M. V. &Lessenius, C. (2012). The role of the tester's knowledge in exploratory software testing. IEEE Transactions on Software Engineering, 39(5).

[47] Bach, J. 2000. "Session-Based Test Management," STQE, 2 (6).

[48] Northrop, L., Clements, P., Bachmann, F., Bergey, J., Chastek, G., Cohen, S., ... & O'Brien, L. (2007). A framework for software product line practice, version 5.0. SEI.–2007–http://www. sei. cmu. edu/productlines/index. html.

[49] Campos, J., Arcuri, A., Fraser, G., & Abreu, R. (2014). Continuous test generation: enhancing continuous integration with automated test generation. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 55-66

[50] Xu, B. (2009, September). Towards high quality software development with extreme programming methodology: practices from real software projects. InManagement and Service Science, 2009. MASS'09. International Conference on (pp. 1-4). IEEE

[51] Fowler, M., &Foemmel, M. (2006). Continuous integration. Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf

[52] Beaumont, O., Bonichon, N., Courtès, L., Hanin, X., &Dolstra, E. (2012, May). Mixed data-parallel scheduling for distributed continuous integration. In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International (pp. 91- 98). IEEE

[53] Bhatti, S. N. (2005). Why quality?: ISO 9126 software quality metrics (Functionality) support by UML suite. ACM SIGSOFT Software Engineering Notes, 30(2), 1-5

[54] Brandtner, M., Giger, E., & Gall, H. (2014). SQA-Mashup: A mashup framework for continuous integration. Information and Software Technology

[55] Jennifer Althouse, Martin Bakal, Paridhi Verma., (2012, August 14) "Continuous Integration in Agile Development." developerWorks., IBM. [Accessed: March 2014.]

[56] Miller, A. (2008, August). A hundred days of continuous integration. In Agile, 2008. AGILE'08. Conference (pp. 289-293). IEEE

[57] Duvall, P. M., Matyas, S., & Glover, A. (2007). Continuous integration: improving software quality and reducing risk. Pearson Education.

[58] Jaakkola, H., &Thalheim, B. (2005). Software Quality and Life Cycles. InADBIS Research Communications

[59] Emelie Engstrom, Per Runeson and Greger Wikstrand, "An empirical evaluation of Regression testing based on fix cache recommendations," 978-0-7695-3990- 4/10 © 2010 IEEE DOI 10.1109/ICST.2010.40

[60] Hendrickson, E., "Agility for Testers", Pacific Northwest Software Quality Conference 2004

[61] L. Crispin and J. Gregory, "Agile Testing: A Practical Guide for Testers and Agile Teams", ISBN-13: 978-0321534460, Edition 1.

[62] Pettichord, B., "Agile Testing Challenges", Pacific Northwest Software Quality Conference 2004.

[63] Amit Juyal, Umesh Kumar Tiwari, Lata Nautiyal, Shashidhar G. Koolagudi, "Agile Plus Comprehensive model for Software Development", In International Journal Computer Technology& Applications, Volume 3 (4), 1378-1383

[64] Bhalerao, S., D. Puntambekar and Ingle, M.,"Generalizing Agile Software Development Life Cycle", In International Journal on Computer Science and Engineering Vol.1(3), 2009, 222-226.

[65] Kohl, J., "Pair Testing. Better Software", Jan 2004

[66] Anita, Naresh Chauhan, “A Framework for Quality Improvement in Distributed Agile Environment”, IEEE International Conference On Research And Development Prospects On Engineering And Technology, March 2013, E. G. S. Pillay Engineering College, Tamilnadu, India

[67] Anita, Naresh Chauhan, “An Object Oriented Design Approach For Modification of Rotten Code Using Regression Testing & Refactoring”, “An Object Oriented Design Approach For Modification of Rotten Code Using Regression Testing & Refactoring” Volume 4, Number 7, 2014, pp, 681-686

[68] Adipat Larprattanakul and TaratipSuwannasart, “An approach for regression test selection using object dependency graph,” 978-0-7695-4988-0/13 © 2013 IEEE DOI 10.1109/INCoS.2013.115

[69] Gerard Meszaros, “Agile regression testing using record and play,” OOPSLA 2003, Oct 26-30, 2003, Anaheim, California. ACM 1-58113-751-6/03/0010

[70] Anita, Naresh Chauhan, “A Regression Test Selection Technique by Optimizing User Stories in an Agile Environment”, 4 th IEEE International Advanced Computing Conference, IACC 2014 (21st -22nd Feb 2014), in ITM University, Gurgaon, India

[71] Anita, Naresh Chauhan, “A Linguistic approach for TCP in an Agile Environment”, 13th Annual International Software Testing Conference (4th - 5 th Dec 2013), Crossing The Chasm: From Assurance To Confirmation, Bangalore, India.

[72] Saff, D. and Ernst, M. D. (2003). Reducing wasted development time via continuous testing. In Fourteenth International Symposium on Software Reliability Engineering, pages 281–292, Denver, CO

[73] Gamma, E. and Beck, K. (2003). Contributing to Eclipse: Principles, Patterns, and Plugins. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA

[74] Saff, D. and Ernst, M. D. (2004). An experimental evaluation of continuous testing during development. In ISSTA 2004, Proceedings of the 2004 International Symposium on Software Testing and Analysis, pages 76–85, Boston, MA, USA

[75] Rady, B. and Coffin, R. (2011). Continuous Testing: with Ruby, Rails, and JavaScript. Pragmatic Bookshelf, 1st edition

[76] M. Virmani. -Understanding devops bridging the gap from continuous integration to continuous delivery. In Innovative Computing Technology (INTECH),2015 Fifth International Conference on, pages 7882, May 2015

[77] Bucchiarone, A., et al. (2018) From Monolithic to Microservices: An Experience Report from the Banking Domain. IEEE Software, 35(3): p. 50-55.

[78] S. Siegel. Object-Oriented Software Testing: A Hierarchical Approach. John Wiley And Sons, 1996

[79] J. Roche. - Adopting devops practices in quality assurance. Commun. ACM, 56(11):38–43, Nov. 2013.

[80] Chen, L. 2015. Towards Architecting for Continuous Delivery. In Proceedings of 12th Working IEEE/IFIP Conference on Software Architecture (WICSA), 131-134

[81] Claps, G. G., Svensson, R. B., Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. Information and Software Technology, 57, 21-31. doi:10.1016/j.infsof.2014.07.009

[82] S. Neely, and S. Stolt, “Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy),” in Agile Conference (AGILE), 2013, pp. 121-128

[83] E. Laukkanen, J. Itkonen, and C. Lassenius, "Problems, causes and solutions when adopting continuous delivery—A systematic literature review,‖ Inf. Softw. Technol., vol. 82, pp. 55–79, 2017

[84] Laukkanen, Eero, Timo OA Lehtinen, Juha Itkonen, Maria Paasivaara, and Casper Lassenius. "Bottom-up adoption of continuous delivery in a stage-gate managed software organization." In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1-10. 2016

[85] Faheem Ullah, Adam Johannes Raft, Mojtaba Shahin, Mansooreh Zahedi, and Muhammad Ali Babar. 2017. Security Support in Continuous Deployment Pipeline. In Proc. 12th International

[86] M.V. Mäntylä, B. Adams, F. Khomh, E. Engström, K. Petersen, On rapid releases and software testing: a case study and a semi-systematic litera- ture review, Empirical Softw. Eng. 20 (5) (2015) 1384–1425, doi: 10.1007/ s10664- 014- 9338- 4

[87] Düllmann, Thomas F., Christina Paule, and André van Hoorn. "Exploiting devops practices for dependable and secure continuous delivery pipelines." In 2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE), pp. 27-30. IEEE, 2018

[88] F. Shull, J. Singer, and D. I. Sjøberg "Guide to Advanced Empirical Software Engineering." Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[89] Kassab, M., DeFranco, J.F., Laplante, P.A.: Software Testing: The State of the Practice. IEEE Software 34(5): 46-52 (2017)

[90] Angara, J.; Prasad, S.; Sridevi, G.. The Factors Driving Testing in DevOps Setting- A Systematic Literature Survey. IJST, Jan. 2017. ISSN 0974 -5645

[91] Clokie, Katrina. "A Practical Guide to Testing in DevOps." (2017).

[92] S. A. I. B. S. Arachchi and I. Perera, “Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management,” no. May, 2018

[93] L. Lwakatare, T. Karvonen, T. Sauvola, P. Kuvaja, H. Olsson, J. Bosch and M. Oivo, “Towards DevOps in the Embedded Systems Domain: Why is It so Hard?,” 2016 49th Hawaii International Conference on System Sciences, 2016.

[94] Mohammad, SikenderMohsienuddin. “Improve Software Quality through practicing DevOps Automation.” SikenderMohsienuddin Mohammad," IMPROVE SOFTWARE QUALITY THROUGH PRACTICING DEVOPS AUTOMATION", International Journal of Creative Research Thoughts (IJCRT), ISSN (2018): 2320-2882

[95] M. Senapathi, J. Buchan and H. Osman, "DevOps Capabilities, Practices, and Challenges: Insights from a Case," EASE'18 Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering, pp. 57-67, 2018
[96] Wiedemann, Anna, and Manuel Wiesche. "Are you ready for Devops? Required skill set for Devops teams." (2018)
[97] Cruzes, D.S., Melsnes, K. and Marczak, S., 2019, July. Testing in a DevOps Era: Perceptions of Testers in Norwegian Organisations. In International Conference on Computational Science and Its Applications (pp. 442-455). Springer, Cham
[98] L. Bass, I. Weber, and Z. Liming, DevOps: A Software Architect's Perspective, Addison-Wesley, 2015
[99] Garousi, V., Felderer, M., Mäntylä, M.V. (2019) Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Information Software Technology 106: 101-121
[100] Sturtevant, D. (2017) Modular Architectures Make You Agile in the Long Run. IEEE Software, 35(1): p. 104-108
[101] Hasselbring, Wilhelm. "Software architecture: Past, present, future." In The Essence of Software Engineering, pp. 169-184. Springer, Cham, 2018
[102] Silva, M. A., Faustino, J., Pereira, R. And Silva, M. M. (2018). Productivity gains of DevOps adoption in an IT team: a case study. In 27th International Conference on Information Systems Development Lund
[103] Taibi, Davide, Valentina Lenarduzzi, and Claus Pahl. "Continuous architecting with microservices and DevOps: a systematic mapping study." In International Conference on Cloud Computing and Services Science, pp. 126-151. Springer, Cham, 2018
[104] A. S. Amaradri and S. B. Nutalapati, Continuous Integration, Deployment and Testing in DevOps Environment. 2016.A. S. Amaradri and S. B
[105] E. Diel, S. Marczak, and D. S. Cruzes,"Communication Challenges and Strategies in Distributed DevOps" 2016 IEEE 11th Int. Conf. Glob. Softw. Eng., pp. 24-28, 2016
[106] Nybom, Kristian, Jens Smeds, and Ivan Porres. "On the impact of mixing responsibilities between devs and ops." In International Conference on Agile Software Development, pp. 131-143. Springer, Cham, 2016
[107] Lwakatare, Lucy Ellen, Terhi Kilamo, Teemu Karvonen, Tanja Sauvola, Ville Heikkilä, Juha Itkonen, Pasi Kuvaja, Tommi Mikkonen, Markku Oivo, and Casper Lassenius. "DevOps in practice: A multiple case study of five companies." Information and Software Technology 114 (2019): 217-230
[108] Sujay Honnamane . Rameshkumar Bar Jumpstarting DevOps with Continuous Testing 2015
[109] Jones, S., Noppen, J. and Lettice, F., 2016, July. Management challenges for DevOps adoption within UK SMEs. In Proceedings of the 2nd International Workshop on quality-aware devops (pp. 7-11)
[110] Chen, Lianping. "Continuous delivery: Overcoming adoption challenges." Journal of Systems and Software 128 (2017): 72-86
[111] Ibrahim, Mahmoud Mohammad Ahmad, Sharifah Mashita Syed-Mohamad, and Mohd Heikal Husin. "Managing quality assurance challenges of DevOps through analytics." In Proceedings of the 2019 8th International Conference on Software and Computer Applications, pp. 194-198. 2019
[112] Wang, Y., Pyhäjärvi, M. and Mäntylä, M.V., 2020. Test Automation Process Improvement in a DevOpsTeam: Experience Report. arXiv preprint arXiv:2004.06381
[113] Leite, Leonardo, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. "A survey of DevOps concepts and challenges." ACM Computing Surveys (CSUR) 52, no. 6 (2019): 1-35
[114] Mascheroni, Maximiliano A., and Emanuel Irrazábal. "Continuous testing and solutions for testing problems in continuous delivery: A systematic literature review." Computación y Sistemas 22, no. 3 (2018): 1009-1038
[115] Surendra Naidu Mullaguru "Changing Scenario of Testing Paradigms Using DevOps–A Comparative Study with Classical Models", Global Journal of Computer Science and Technology. 2015
[116] Virmani, M., 2015, May. Understanding DevOps and bridging the gap from continuous integration to continuous delivery. In Fifth International Conference on the Innovative Computing Technology (INTECH 2015) (pp. 78-82). IEEE
[117] Hemant Madaan, (2023).Continuous Testing: The Key to Quality Assurance in the DevOps Era.https://devops.com/continuous-testing-the-key-to-quality-assurance-in-the-devops-era/

Author Profile

Amit Bhanushali is a highly accomplished software quality assurance professional with over 22 years of experience in the IT industry. He earned his Master's in Business Data Analytics from West Virginia University in 2017. Based in West Virginia, USA, Mr. Bhanushali is a Senior IEEE Member and has significantly contributed to software testing research and practice. His expertise spans automation testing, performance testing, DevOps, and CI/CD implementation. He has also led testing efforts in complex cloud environments. In addition to testing, Mr. Bhanushali has authored several articles exploring cutting-edge topics like artificial intelligence and machine learning. His published research demonstrates his thought leadership and impact on software quality engineering. Mr. Bhanushali's accomplishments have been recognized through prestigious appointments. He serves as a reviewer for the Elsevier journal and has been a hackathon judge. His contributions were further honored in 2023 when he received the International Achievers' Award. With his sustained record of excellence across software development, testing, and research, Mr. Bhanushali continues to be an influential leader in his field. **OCR ID**0009-0005-3358-1299