International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.8, Issue 4, July-August 2025|

DOI: 10.15662/IJSRAT.2025.0804007

Cloud-Native Shift Left Accessibility Engineering for Scalable Agile Product Development Ecosystems

Dr. Abhishek Raghuvanshi

Professor, Department of CSE, Mahakal Institute of Technology and Management, Ujjain, MP, India

ABSTRACT: In software development circles, the “cloud-native” end-to-end product development or agile software development lifecycle approaches have come into prominence, with making sure products are accessible and available from the very beginning of the SDLC process. This research paper proposes a Shift-Left Accessibility Engineering Framework for agile product development eco-systems to bring accessibility requirements, accessibility automated testing, inclusive design validation, developer enablement and accessibility monitoring all under one roof in the cloud-native DevOps pipelines to support the scalability of agile product development eco-systems. The table below shows that the framework correlates the accessibility checkpoints with backlog refinement, user story writing, design prototyping, coding, continuous integration/continuous deployment, quality assurance and post-release analytics. The study illustrates a combination of standards based accessibility criteria (WCAG), automated accessibility scanning, containerized test environment, design system governance, and cross functional working in an agile manner to reduce remediation costs, increase user inclusivity, and boost product quality at scale. The overall emphasis of the paper is accessibility engineering is also another responsibility which should be shared by the product owners, designers, the developers, the testers, DevOps and compliance stakeholders. It also focuses on the use of cloud-native architectural principles, microservices, reusable components and observability tools to ensure ‘continuous accessibility assurance’ for any product in an ecosystem of distributed products. With the proposed practice, it provides an effective model in which accessibility is also an intrinsic part of the agile process for development and deployment, but does not affect delivery velocity. To conclude, the study highlights the importance of accessibility accessibility left in transitioning away from a reactive approach to digital product development, solely focused on compliance, towards a proactive and inclusive approach; and to promote more sustainable practices.

.

KEYWORDS: Cloud-native development; Shift left accessibility; Accessibility engineering; Agile product development; DevOps pipeline; Inclusive design; WCAG compliance.

I. INTRODUCTION

This growth and ubiquity of digital products has had an impact on software development and the software release process, which has turned into a continuous process, with an emphasis on users and empowered by the cloud. Adopting agile methods, DevOps automation, microservices, containerization, and cloud native infrastructure, all modern-day organizations are building, deploying, monitoring and optimizing applications. Cloud-native technologies provide the best ways to build applications to scale in the "wild world of public, private, hybrid cloud" like containers, microservices, service meshes and immutable infrastructure and declarative APIs. These practices have passed on to us a better delivery both in terms of speed, scale, resilience and operational flexibility – but along with the upside come new challenges in the areas of quality, compliance, user experience and digital inclusion. Of these, accessibility is now considered to be an engineering issue and no longer a secondary design issue.

In the software development world, Accessibility is defined as making the software "friendly" to computer users of all abilities which includes visual, auditory, cognitive, motor, verbal, neurological as well as learning disabilities. The Web Content Accessibility Guidelines (WCAG) help with making content more accessible using technology-independent (and technology testable) criteria:tható: Perceivable: Ensure that content is presentable to users in ways they can perceive.WCAG: Web Content Accessibility Guidelines to make content more accessible using technology-independent (and technology testable) criteria: Percivable: Ensure content is presentable to users in ways they can perceive. Yet, even with readiness of accessibility criteria and accessibility instruments, quite a few groups continue to consider accessibility as a last action in the compliance process, close to the launch or because of customer complaints, an accessibility audit or legal risk. A late approach tends to take more time for remediation, redesign, resulting in

International Journal of Science, Research and Technology (IJSRAT)
|Bimonthly | Peer-Reviewed | Scholarly Journal|
|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.8, Issue 4, July-August 2025|
DOI: 10.15662/IJSRAT.2025.0804007

disjointed user experiences/compatibility and accessibility problems that are difficult to resolve in the product architecture defined early on [1].

The problem can be resolved with the idea of "shift left" which involves moving quality, security, testing and compliance activities earlier in the software development lifecycle. By incorporating thinking about accessibility at the initial product planning stages, requirement analysis, design, development and CI, shift left promotes good accessibility by incorporating it all the way across the software development lifecycle. A hot topic for large companies when dealing with agile product development, where features are released slowly and often, is especially appropriate. If not addressed at least at backlog refinement, a product design system, code review, automatic testing and deployment pipelines (or anywhere between) then accessibility defects can mount up with each sprint and can be systemic throughout sprints and throughout the product ecosystem [2] [3].

In cloud-native environments, it can be both easier and more vital to make a positive security incident into a positive opportunity. Cloud-native however are generally distributed, modular and in a continuous state of change, which leads to the possibility of service, interface, component and platform (application) availability in the system changing. There's the possibility of one single reusable part having access issues that could impact multiple applications, user flows, and/or customer facing services. On the other hand, cloud-based software can offer businesses the chance of automation, which can help guarantee ongoing access validation. Teams can find and stop accessibility defects from even happening by using containerized testing environments, CI/CD pipelines, design tokens, re-usable component libraries, infrastructure as code (IaC), observability tools, and automated scanning. Therefore, it is critical to understand that accessibility isn't just a design concern in cloud native environments, it's an Engineering, Governance & Operations concern.

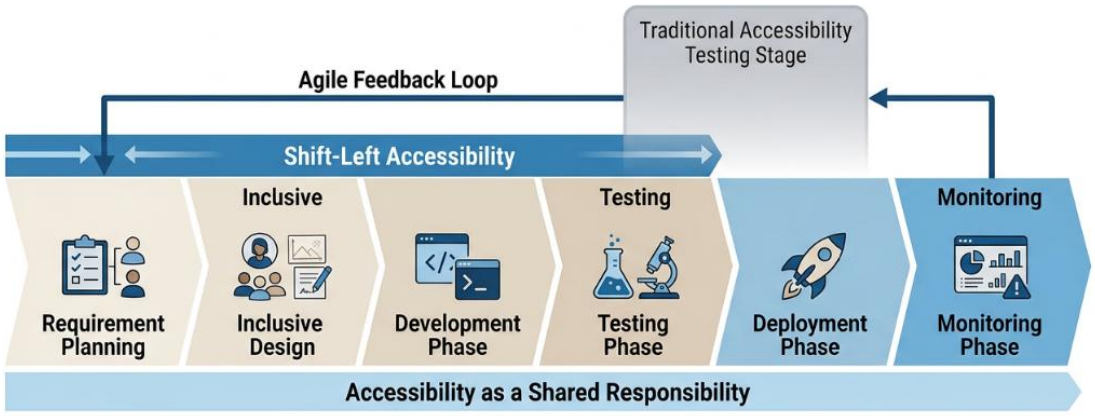

Figure 1: Conceptual Overview of Cloud-Native Shift Left Accessibility Engineering

This research paper 'Cloud-Native Shift Left Accessibility Engineering for Scalable Agile Product Development Ecosystems' looks at the necessity of incorporating accessibility engineering via a structured and scalable framework into cloud-native agile workflows. According to the proposed framework, accessibility should be included as part of the loops during the following seven phases/layer in the lifecycle: accessibility-aware requirement engineering, accessibility in the developer practices, built-in accessibility testing by means of the CI/CD pipelines, accessibility-focused developer practices, cross-functional agile QA testing, user storytelling as a part of accessibility, and post-release accessibility observability. This "stack" enables the framework to connect these layers and operationalise these tacit outcomes – taking accessibility out of a "reactive 'audit' process" and into the capability of engineering.

It starts with the requirements and planning phase: When writing a vision, user persona, requirement of done and acceptance criteria, you must put accessibility in them. Besides the functional requirements, it's important to include accessibility requirements such as keyboard navigation, semantically, how compatible the User Story is with the screen readers, sufficient contrast during the representation, error identification and focus management and accessible authentication. WCAG 2.2 Low Impact High Reward spent time on trying to identify new success criteria which would risk the validation of WCAG 2.1 becoming a W3C Recommendation, but went this direction to underscore the need for new accessibility practices in today's digital products. An access criteria on a backlog level can help teams create

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.8, Issue 4, July-August 2025|

DOI: 10.15662/IJSRAT.2025.0804007

accessible functionality alongside they estimate, design, develop and test and lead them to consider accessibility instead of compliance as an external issue.

The framework emphasizes on accessibility enabled design systems and adaptable components in design and development. Most agile enterprises that operate in the cloud follow a common UI library and component driven front end in their product teams. These common elements – if they are part of the default environment – can be ports of call for everyone. But on the other hand, not attainable rounded sections can broadcast mistakes routinely with swiftness. It should also, therefore, be part of a design system's governance to ensure that accessibility is part of the design system documentation, that usage rules involve accessibility, that patterns for design and keyboard interaction are identified, that (if necessary) ARIA is used for accessibility, and that automated runs are done when components are released to design systems.
On the testing/deployment tier, it includes: Automated accessibility testing, marking by accessibility experts, assistive technology testing and validation with users in the CI/CD pipeline. Common issues to check for are missing alt text, insufficient color contrasts, poor headings structure, issue with form labels and/or incorrect labeling of ARIA. Nevertheless, with automation alone it is still not possible to achieve full accessibility as many accessibility problems only can be determined with human judgment and/or contextual understanding and testing with assistive technologies. Thus, the suggested framework comprises of both automated validation and manual testing, inclusive design review and feedback from people with disabilities.

Last but not least, they include accessibility in post-release monitoring and constant improvement. Cloud native observability is universal in populating systems to monitor the performance, reliability and security. A similar thought process can be used for accessibility as well: Over a time period, the number of accessibility-defect entities that were identified by accessibility-auditing results, user feedback, number of support tickets, and trends of regressions over product releases can be a good indicator of accessibility. Their continuing lack of accessibility creates an opportunity for organisations to over time gain a better understanding of what they can do to make their designs more accessible. To conclude, this piece proposes that moving left accessibility engineering is essential to designing inclusive, scalable and sustainable digital products in Agile development cycles, and delivers best results as a “cloud first” strategy.

II. RELATED WORK

The modern design of digital products is modified, deployed and maintained by cloud native software development. Saez gave an overview of the distinction between cloud-based and cloud native - she commented that “cloud native” is not necessarily “cloud hosted applications”, but rather building apps to be cloud-enabled to be cloud-scalable, cloud-resilient, automatic, and continuous delivery.” It also is significant in the context of accessibility engineering because to be accessible, systems must support accessibility across the entire distributed architecture, across the reusable services and throughout the system during the automated process of its delivery to the deployed system and continue to do so in the deployed system.

The "left shift of accessibility" in agile product development lifecycle was explicitly addressed by Dr Suvvari [2]. The study underscored the need for incorporating the accessible features at the outset of planning, design, development and not through a process of testing/compliance at a latter end. This work is well aligned with the ongoing research because it supports the theory being proposed that accessibility should be a proactive measure and approach to take in engineering. Yet, although Suvvari talked mostly about accessibility in agile development, the current paper takes this concept further into a cloud-native environment where, along with accessibility, scalability and automation, DevOps and continuous monitoring also are critical.

Ebert and Paasivaara talked about the barriers to scaling agile in big companies, as well as the practices [3]. Their work is relevant, as access issues in large, complex product ecosystems can no longer be left to just individual engineer awareness, and manual audits. The issues of accessibility should be tackled through regular business, governance, re-usables and common quality criteria instead. Calefato and Ebert, similarly, conducted research about cooperation in agile teams in a distributed environment and showed that communication, coordination and common tools are important in a geography-distributed software development environment [4]. It's significant in terms of being able to reach cloud customers because many agile teams are distributed, span time zones and technical areas.

Pargaonkar has compared two models of software development life cycle (Sudo Agile and Sudo Waterfall) and briefly enlisted the advantages, limitations and suitability of the same with software quality engineering [5]. This comparison even more reinforces the need of shift left approach, which means that the traditional lifecycle models might be causing

International Journal of Science, Research and Technology (IJSRAT)
|Bimonthly | Peer-Reviewed | Scholarly Journal|
|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.8, Issue 4, July-August 2025|
DOI: 10.15662/IJSRAT.2025.0804007

issues 'late in the cycle', whereas agile models and practices provide opportunities for continuous validation and timely feedback. The Agile Manifesto includes a set of the foundational principles of iterative development, customer collaboration, working software and responsiveness to change [6]. Such principles have led to the use for accessibility engineering as they allow accessibility requirements and user feedback to be added on an ongoing basis in the product development process.

T. Ozkan and P. Mishra have discussed the various tools used in agile project management and how they help to maintain the visibility, tracking and collaboration aspect of the agile workflow [7]. These tools can be helpful for associating accessibility tasks, defect, acceptance criteria etc. to the sprint level responsibilities. Alaidaros, Omar and Romli highlighted the Kanban practices, challenges and opportunities, which illustrated the usefulness of the visual workflow management and continuous improvement [8]. Accessibility backlog management and defect prioritization can be done using Kanban-based practices.

But already during the COVID-19 pandemic, Schmidtner, Doering and Timinger have been investigating the agile working, on which they focus on remote collaboration and agile working as adaptive working [9]. Drutchas and Eppinger [10] evidence that supported path agile principles could be adapted and applied outside of regular software environment led to the downgrade of agile suggestion to the product creating process of hardware and software complex products. Last but not least, the following access hub-and-spoke model of tooling for integrated accessibility in distributed development was proposed by Calefato and Lanubile, which emphasizes the need of tooling for integrated accessibility in distributed CI/CD systems.
Finally, Calefato and Lanubile provide the following hub-and-spoke idea of how to do tooling for integrated accessibility in distributed development. Collectively, these studies provide a foundation for a model which can incorporate cloud native, agile distributed and shift left accessibility engineering.

III. CLOUD-NATIVE SHIFT LEFT ACCESSIBILITY ENGINEERING FRAMEWORK

1. Overview of the Framework

Cloud-Native Shift Left Accessibility Engineering Framework (ASLAF2) aims to seamlessly embed accessibility principles into the very beginning of agile product development and make them a part of cloud-native delivery. Likewise, rather than simply a compliance exercise at the end of the product development process it has ushered in the concept of accessibility as a continuous commitment to engineering that includes a variety of stakeholders from product owners to UX designers, product developers, testers, DevOps to accessibility consultants, and organisational business stakeholders. In today's agilely managed systems – where products are developed in short increments of time, by using reusable components, microservices, and continuously releasing and monitoring – accessibility needs to be incorporated into all the planning, designing, development, testing, release, and monitoring.

This framework has at its core, attempting to proactively stop accessibility defects from occurring or becoming too costly and complicated to fix. In general, traditional accessibility methods consist of an audit at the end of the development process when a design, coding structure and user flows are defined. This reactive approach is costly when it comes to remediation and cycles are not streamlined. By contrast, an attitude shift moves the process of making decisions about accessibility and the process of validating activities closer to the beginning of the software lifecycle. This early integration is even more important when dealing in a cloud-native Application world where Apps are continuously appened, scaled and deployed across multiple environment and to multiple user groups.

International Journal of Science, Research and Technology (IJSRAT)
|Bimonthly | Peer-Reviewed | Scholarly Journal|
|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.8, Issue 4, July-August 2025|
DOI: 10.15662/IJSRAT.2025.0804007

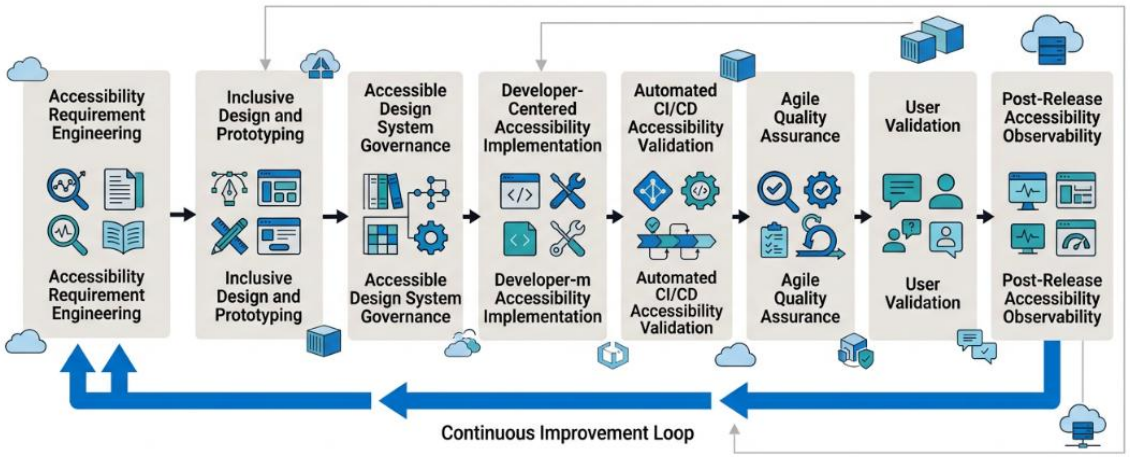

Figure 2: Cloud-Native Shift Left Accessibility Engineering Framework

2. Accessibility Requirement Engineering

Accessibility requirement engineering is the initial layer in the framework. In this layer, accessibility is taken into consideration both in the process of defining product vision and when creating the backlog and sprint planning as well as analysing the requirement. Whenever you're talking about a product, and you're trying to make things easier for people to access it, it's best to think about it at the beginning, when you're big-picture thinking and the product owner and business analyst are involved, rather than when the product is developed. Don't put goals into accessibility as a separate activity at the end of the coding, they belong in epics, user stories, acceptance criteria and definition of done.

For example, if the user story is for the login screen, then the user story shouldn't only include how the user is supposed to log-in or the authentication sequence, it should also include the description of how the user enters the username and password. It should also cover accessibility of the form and input fields , clear and accessible error messages and the ability for screen readers to detect the focus and indicate it. Early identification of such needs will allow the designers and developers to consider accessibility the plan from the start. This minimises the chances of misunderstandings and helps prevent accessibility not being taken into consideration when applying and performing sprints.

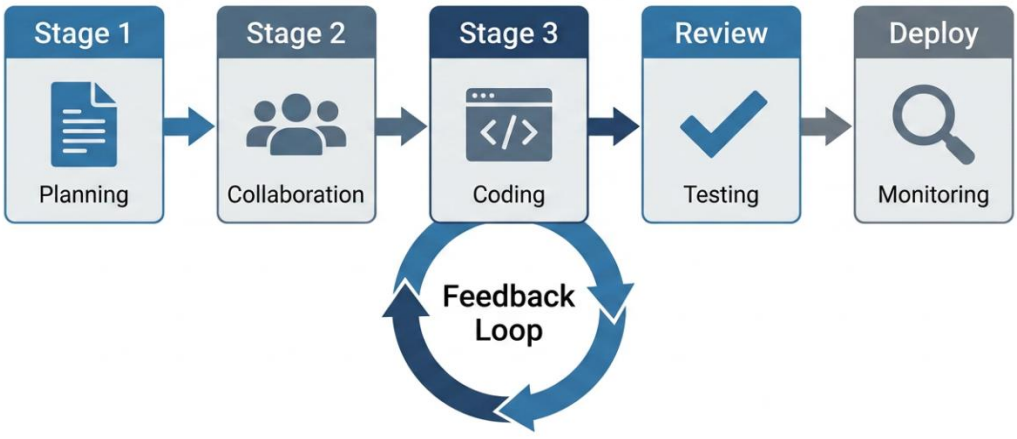

Figure 3: Accessibility Integration Across Agile Sprint Lifecycle

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.8, Issue 4, July-August 2025|

DOI: 10.15662/IJSRAT.2025.0804007

3. Inclusive Design and Prototyping

The second part of the framework is that of inclusive design and prototyping. During this phase, teams that are committed to accessibility work with UX/UI to create accessible user flows, wireframes, visual design and interactions with the user. Inclusive design is used to develop a digital solution designed for everybody, including all users, physical or sensory health, digital literacy or background in media or languages, as well as the type of digital tool they use to access the solution. So there should be a relationship between the accessibility and subjects of colour contrast, Typography, organisation of Navigation, hierarchy of content, design of the forms, prevention of error and size of the target and feedback.

Accessibility should be built up into the design prototypes, before handing to their developers. Examples of items under review may include visual logic—flow through a passage— easy identification of interactive controls, clarity of instructions, and ability to perform without an over dependence on colour or sounds or pointing and clicking. These reviews can be part of a sprint refinement, design review and/or prototype validation session in agile teams. Therefore, accessibility isn't just one of the boxes that the designer is required to check.

4. Accessible Component and Design System Governance

Participatory component and design system management is the third one available. Common product ecosystems in cloud-based systems consist of pannable components that are re-used for the UI, shared front end libraries and a style guide. Such systems facilitate teams to maintain consistency and speed up its products delivery. But they also have a direct impact, on scalability of accessibility. Any product that utilizes a shared component like a navigation menu, table, alert, dropdown or modal will be finding value in it if it is accessible by default. If it is not possible to reach the figure of the defect, it can be spread over a number of applications.

The proposed framework will thus include the validation of accessibility for every part that can be reused in the design system to guarantee that accessibility is not negatively affected. For each component, a description of accessibility documentation, keyboard behaviour, semantic structure, responsive behaviour, focus management rules, error handling patterns and considerations for AT. Finally, a design system must not only define the visual appearance of the components, it must define how they will behave on the screen to users using a screen reader, magnifier, voice input, keyboard or any other assistive technology to find and interact with the screen. A governance model like this enables accessibility to scale across all products without having to become a 'solution-a-thon' of tackling the same accessibility issues repeatedly by every team.

5. Developer-Centered Accessibility Implementation

The fourth level of the framework aims at developer techniques and practices. Developers are key in making accessible working software based on accessible designs. Even if it is a design that looks inclusive, when poorly executed it may be an obstacle to users. This layer thus focuses on semantic coding, accessibility aware development practices as well as peer review and developer enablement. Use of native HTML elements needs to be eased, headings structure should be supported, appropriate labels should be provided, the order of focusing should be logical, keyboard access should be supported and developers need to refrain from misuse of ARIA and duplicating of ARIA.

Another aspect that needs to be encouraged in code with review culture is accessibility. Reviewers should check the following attributes of interactive controls: Are interactive parts of the web page visually accessible without a mouse; Are form controls properly associated with the form labels; Are visually dynamic changes to the web page communicated to Assistive Technology; Do custom interactive controls function as intended? It's important to train developers because many of the accessibility bugs are a result of a lack of awareness. This encourages accessibility to be part of engineering quality by putting accessibility into their regular practice of development.

6. Automated Accessibility Validation in CI/CD Pipelines

To make matters worse, there is automated accessibility validation at the continuous integration and continuous deployment stage (fifth layer). Automation is the base for speedy development, testing, deployment, and monitoring of software to support cloud-native development. Common accessibility problems at the start and often can also be automated. These automated accessibility tools can be integrated into automated testing efforts, such as end-to-end automated testing, integration tests, unit tests and pull requests. These can detect problems like a lack of alternative text, a lack of good color contrast, the lack of labels on forms, duplicate IDs, empty links, errors in ARIA attributes, and structural markup errors.

International Journal of Science, Research and Technology (IJSRAT)
|Bimonthly | Peer-Reviewed | Scholarly Journal|
|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.8, Issue 4, July-August 2025|
DOI: 10.15662/IJSRAT.2025.0804007

The containerised test environments make testing in a consistent and consistent manner to flow from the development to staging and production like environment easily accessible. Pipeline rules also can be set to prevent releases if there are critical accessibility violations. This offers an automatic quality gate to keep serious defect(s) from progressing. However, it's important to not view Automated Validation as completely resolving the situation, just as a component. While automated tools can be helpful for finding the repetitive and rule based problems, they do not assess usability, clarity, context or entire experience for users with disabilities.

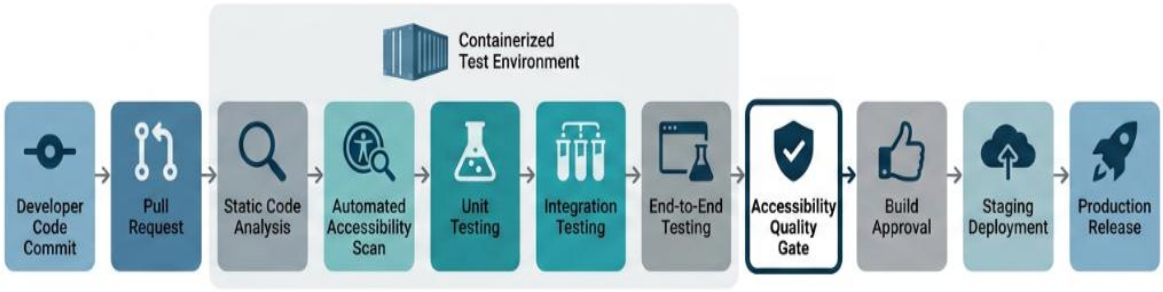

Figure 4: CI/CD Pipeline for Automated Accessibility Validation

7. Agile Quality Assurance and User Validation

The 6th layer is called Agile quality, user validation. Manual should be the main method of testing as even the most powerful automated tools will not be able to solve all accessibility issues. Tests that are done by the QA teams should include: Keyboard-only tests, Screen reader tests, Browser zoom tests, Responsive layout tests, Focus order review, validation of forms, and error recovery tests. These practices can provide assurance that the automation process will not ignore features that may interfere with automated reading, including drawbacks like navigating next to the main content or reading content out of order, dynamically generated content which is not clearly accessible, unclear instructions, etc., or that there are ways to interact that are not appropriate for technology supports.

Sprint level testing, and product validation and review, should involve accessibility validation. Specially trained access testers/expert accessibility specialists, or accessibility developers, should be involved in the review session when high impact user journeys are created. Wherever possible an attempt will be made to involve user(s) with disabilities in a usability test to identify practical issues that may not be picked-up in a technical check. With this layer, access also is taken into consideration beyond just access to standards – as a user experience.

IV. PERFORMANCE EVALUATION

1. Evaluation Overview

Cloud-Native Shift Left Accessibility Engineering Framework's performance measures include whether accessibility practices are successfully embedded into agile and cloud-native product development initiatives in the form of effectiveness within the process. The goal of the framework is to minimize late stage remediation and increase “inclusive product quality;” as such, the effectiveness of the framework must be judged and evaluated not just by the results of the technical tests, but by the efficiency of the process, the defects that are eliminated, delivery impact and the increase in user experience. Evaluation thus takes into account not only the criterion of accessibility compliance, but also the criterion of effectiveness of automation controls, a criterion of defect management and post-release accessibility stability, as well as accessibility, and development velocity.

2. Accessibility Defect Detection and Reduction

It is also one of the key measures of framework performance and defects in accessibility can be fixed during the lifecycle of the software development. In a traditional way, the majority of the accessibility problems are found either in the final test or in an external audit. Defects would be found at an earlier stage under the proposed framework as part of the review of requirements, design validation, code review and as part of the testing process on CI/CD pipelines. The number of accessibility defects can be used to measure the performance, based on the comparison between the number of accessibility defects which are detected in the development phase and which are detected in the post-release phase. If

International Journal of Science, Research and Technology (IJSRAT)
|Bimonthly | Peer-Reviewed | Scholarly Journal|
|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.8, Issue 4, July-August 2025|
DOI: 10.15662/IJSRAT.2025.0804007

successful, there will be an increase in percentage of defect discovery at early stage, and a decrease in number of critical accessibility defect at production.

3. CI/CD Automation Effectiveness
All of the framework relies on automation of the accessibility validation process in cloud native CI/CD pipelines. So, it's a crucial assessment area, and it's effectiveness of automation that exists. Such measurements involve how many automated scans are run during a sprint, how many issues automatic scans are flagging up, false positive rate and compliance time to address the issues identified by the automatic scans. Automated testing is also a way to establish how well certain accessibility problems are avoided from entering into staging or production environments. The successful automation process should lead to more consistency, less time to manually test the parts required and a fast response to the developer without slowing deployment.

4. Agile Delivery and Remediation Efficiency
A crucial one is how it will impact agile delivery performance during Accessibility integration. There is no need for delays due to the framework while running the sprints; it's intended so there won't be any reseed of the same problems if they should crop up again later in the sprint. Some evaluation metrics might be: average remediation time, or average sprints carry over as a result of accessibility defect, or effort/cost of remediation defect at various stages. As accessibility is moved to the left, defects found in design and/or coding processes should be easier to fix than post-release defects. This makes the team more productive and help them towards continuous delivery.

5. User Experience and Compliance Outcomes
The last measurement criterion is related to the quality of user experience and compliance. They want to demonstrate tangible improvements that allow them to meet accessibility standard, usability that considers people using AT, and a decrease in people's AT support complaints in line with the framework. Manual testing, screen reader validation and keyboard navigational checks can measure real world accessibility performance, or feedback sessions with users can. Recurring problems of accessibility and regression patterns should also be a part of the post-release monitoring.

V. CONCLUSION AND FUTURE WORK

1. Conclusion
In this research paper, we present the framework of a Cloud-Native Shift Left Accessibility Engineering with respect to Scalable Agile Product Development Ecosystems. Moving toward the observation that accessibility can no longer be considered a "last minute" compliance issue, but one that rests on the shoulders of software engineer(s) at every phase of the software lifecycle. Organizations can mitigate late stage remediation efforts, enhance the quality of the product, and deliver more inclusive digital experiences, by democratizing accessibility practices earlier in the requirement analysis, user story creation, designing, development, testing and deployment of a product.

The proposed framework addresses incorporating accessibility into agile, cloud-native flows in the requirements engineering, inclusive design, dev implementation, CI/CD automated validation, quality-assurance agile process and post-release observability. Through this structured process, everyone from product owners to designers, developers to testers, DevOps engineers to accessibility experts and on to organizational leaders, collaborate. It also evolves accessibility to the latest practices for the cloud-based environment: Automation, re-used components, continuous delivery and monitoring. The general vision of the framework helps organizations move away from reactive-approach accessibility fixes and towards a more proactive accessibility engineering way of preventing issues.

VI. FUTURE WORK

The future work can continue the present research by implementing the presented framework in other contexts such as organizations and industry with validations to prove the framework. Empirical research can be carried out in software companies, public digital service platforms, educational technology systems, healthcare applications and financial technology products, among others, for example, to quantify the utility of the framework or how well it works in practice. Future work: To provide a more holistic understanding of the benefits of shift left accessibility activities, it is recommended that the presence of a correlation between improvements in accessibility defect reduction, remediation cost, release efficiency and user satisfaction be assessed both prior to and following accessibility improvements.

Other areas of significance where future research is important for the Accessibility engineering include AI and ML. Using AI tools automated remediation might be possible, accessible code suggestions might also be provided, high-risk

|Bimonthly | Peer-Reviewed | Scholarly Journal|
|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.8, Issue 4, July-August 2025|
DOI: 10.15662/IJSRAT.2025.0804007

components might also be predicted and accessibility defects may be identified. But, the latter tools need to be thoroughly assessed to check its accuracy and reliability. Additionally, future research will involve dashboards of accessibility observability, accessibility maturity assessment models and accessibility organization-wide governance metrics. These extensions would extend the framework and make accessibility engineering be more scalable, measurable, and sustainable in a cloud native agile world.

REFERENCES

[1] A. Saez, "Cloud-based versus cloud-native: what's the difference?," *Cloud Native Computing Foundation*, Sep. 4, 2023. [Online]. Available: https://www.cncf.io/blog/2023/09/04/cloud-based-versus-cloud-native-whats-the-difference/
[2] S. K. Suvvari, "Shift left: Moving the inclusion of accessibility functionalities to the left in agile product development life cycle," *Journal of Computational Analysis and Applications*, vol. 31, no. 4, 2023.
[3] C. Ebert and M. Paasivaara, "Scaling agile," *IEEE Software*, vol. 34, no. 6, pp. 98–103, 2017.
[4] F. Calefato and C. Ebert, "Agile collaboration for distributed teams," *IEEE Software*, vol. 36, no. 1, pp. 72–78, 2019, doi: 10.1109/MS.2018.2874668.
[5] S. Pargaonkar, "A comprehensive research analysis of software development life cycle (SDLC) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering," *International Journal of Scientific and Research Publications*, vol. 13, no. 8, pp. 120–124, 2023, doi: 10.29322/IJSRP.13.08.2023.p14015.
[6] K. Beck *et al*., "Manifesto for agile software development," 2001. [Online]. Available: http://agilemanifesto.org/
[7] D. Ozkan and A. Mishra, "Agile project management tools: A brief comparative view," *Cybernetics and Information Technologies*, vol. 19, no. 4, pp. 17–25, 2019.
[8] H. Alaidaros, M. Omar, and R. Romli, "The state of the art of agile Kanban method: Challenges and opportunities," *Independent Journal of Management & Production*, vol. 12, no. 8, pp. 2535–2550, 2021.
[9] M. Schmidtner, C. Doering, and H. Timinger, "Agile working during COVID-19 pandemic," *IEEE Engineering Management Review*, vol. 49, no. 2, pp. 18–32, 2021, doi: 10.1109/EMR.2021.3069940.
[10] J. Drutchas and S. Eppinger, "Guidance on application of agile in combined hardware and software development projects," *Proceedings of the Design Society*, vol. 2, pp. 151–160, 2022, doi: 10.1017/pds.2022.16.
[11] F. Calefato and F. Lanubile, "A hub-and-spoke model for tool integration in distributed development," in *Proc. 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, 2016, pp. 129–133.