Evaluation of Testing Tools for Assessing the Accessibility of Android Apps in Accordance with EN 301 549

Makhabat Usenalieva, Thomas Franke, Rainer Wasinger
Department of Computer Science
University of Applied Sciences Zwickau,
08056 Zwickau, Germany
e-mail: {makhabat.usenalieva.mj1 | thomas.franke | rainer.wasinger}@whz.de

***Abstract*—European Union accessibility legislation now requires both public- and private- sector organisations to meet defined accessibility standards for mobile applications, with non-compliance risking the exclusion of users and potential legal consequences. The primary objective of this work is to evaluate four freely available accessibility testing tools for native Android development with respect to their coverage of accessibility standards, accuracy, ease of integration into development toolchains, and developer usability. This is achieved by 1) deriving a practical set of accessibility requirements from EN 301 549 to test the accessibility of native Android apps, 2) creating a prototypical testbed application called *Accessibility Lab* to support reproducible evaluation results, and 3) using the testbed to evaluate the four tools. Results indicate that while automated tools support early-stage code-level validation, they provide only limited coverage for runtime usability issues. Manual screen reader testing remains essential for identifying critical usability barriers. This work highlights the complementary strengths of the evaluated automated, semi-automated, and manual methods and provides structured guidance on the effective integration of freely available accessibility tools into Android development workflows.**

Keywords-D.2.5.t Usability testing; J.9 Mobile applications; K.4.2.b Assistive technologies for persons with disabilities.

I. Introduction

Native Android applications play a prominent role in digital ecosystems, supporting a wide range of everyday tasks. In Europe, Android accounts for over 60% of the mobile operating system market as of January 2026 [1], making it a critical platform for user access to digital services.

Ensuring accessibility in these applications is essential given the widespread prevalence of sensory impairments. According to the World Health Organization, approximately 90 million people in the broader European Region experience vision impairment or blindness, and 190 million live with hearing loss or deafness [2]. Accessibility also supports users with temporary impairments and enhances general usability [3]. Applications that fail to meet accessibility requirements risk excluding users and violating legal mandates.

In the European Union, mobile apps are subject to formal accessibility obligations. For development teams, systematically validating conformance with these requirements is challenging, not only because the process is time-consuming and resource-intensive, but also because most developers are not specialists in accessibility guidelines. Evolving Android development practices and frameworks further complicate testing workflows, as teams must combine automated tools, semi-automated checks, and manual evaluation to achieve adequate coverage of both technical criteria and real-world usability.

To address these challenges, this work evaluates the capabilities of four freely available tools for assessing the accessibility of native Android applications. The analysis focuses on tools suitable for contemporary Kotlin-based development and declarative user interfaces based on Jetpack Compose. To support this evaluation, a demonstrator application, *Accessibility Lab*, was created. This demonstrator intentionally combines accessibility-compliant features with typical accessibility violations, enabling systematic, repeatable comparisons of tool coverage, accuracy, and practical usefulness from a developer's perspective.

The remainder of this paper is organised as follows: Section II discusses related work. Section III outlines the methodology. This is followed by a presentation of the results in Section IV and limitations in Section V. Section VI then presents the conclusion and topics for future work.

II. Related Work

The legal foundation for digital accessibility in the European Union is primarily provided by the Web Accessibility Directive (EU) 2016/2102 and the European Accessibility Act Directive (EU) 2019/882. While both instruments aim to improve accessibility for persons with disabilities, the Web Accessibility Directive focuses on online content and mobile applications of public-sector bodies (e.g., federal and state ministries, public universities, schools, and hospitals), whereas the European Accessibility Act Directive addresses a broader set of products and services offered mainly by small, medium, and large (but not micro-sized) private-sector operators [4][5]. The technical implementation of these directives is supported by the European Telecommunications Standards Institute's (ETSI) harmonised European Standard for Accessibility Requirements for ICT Products and Services, called EN 301 549 [6]. This standard incorporates the Web Content Accessibility Guidelines (WCAG) 2.1 Level AA and extends it with additional provisions specific to software and mobile platforms [7].

Although EN 301 549 explicitly includes accessibility requirements for native mobile applications, researchers have identified challenges in operationalizing its requirements in Android environments. Seixas Pereira *et al.* report interpretive inconsistencies and a lack of concrete implementation guidance [8]. Olsson and Zubenko emphasize technical barriers stemming

from dynamic User Interface (UI) rendering and layout variability on mobile devices [9]. These findings indicate that direct adaptation of web-based accessibility standards is insufficient for native Android applications.

Beyond standard interpretation, prior studies have documented recurring accessibility barriers across mobile platforms. Frequent issues include missing text alternatives, weak semantic hierarchies, insufficient contrast, and limited assistive technology support [7][8][10]. These problems are exacerbated by the adoption of Jetpack Compose, a declarative UI framework that generates layouts at runtime. While Jetpack Compose simplifies development, its rendering model complicates automated accessibility checks and can limit the effectiveness of existing evaluation tools [11]. Low developer awareness and inconsistent adoption practices further impede accessibility compliance [3].

Prior work distinguishes between expert inspection, checklist based review, automated tool support, and user testing as complementary approaches to accessibility evaluation. In the Android context, tools such as the Accessibility Scanner, Android Lint, the Accessibility API, and the Accessibility Test Framework (ATF) support varying degrees of automated and semi-automated analysis [12]–[15]. However, existing studies offer only partial comparisons of these tools and rarely relate their capabilities to specific accessibility standards.

This study extends prior work by providing a structured comparison of freely available Android accessibility evaluation tools. The analysis relates tool behaviour to selected EN 301 549 requirements and examines how these tools integrate into contemporary Android development workflows, addressing gaps in the literature on accessibility testing for declarative user interface frameworks.

III. Methodology

This work applies a structured evaluation strategy to assess freely available Android accessibility testing tools. The approach involved (A) deriving concrete evaluation criteria from EN 301 549, (B) embedding those criteria into a controlled testbed application called *Accessibility Lab*, and (C) systematically applying accessibility tools to the testbed application to detect known accessibility issues.

A. Derivation of Evaluation Test Criteria

The evaluation began with the development of a concise set of testable criteria based on EN 301 549, focusing specifically on Section 11 (Software). Accessibility guidelines from Di Gregorio *et al.* [3] served as an initial reference. From these guidelines, only requirements applicable to native Android environments were retained. Items without a clear mapping to EN 301 549 provisions were adapted or removed to maintain normative consistency.

A secondary review of EN 301 549 identified additional relevant requirements, including assistive technology interoperability, gesture-independent control, and programmatic detection of dynamic state changes. Where applicable, these provisions were integrated into the final set of criteria. Table I shows the resulting coverage categories that provided the foundation for the *Accessibility Lab* prototype and tool assessment.

B. Prototype Accessibility Lab Application

To support reproducible evaluation, a Jetpack Compose based prototype application, *Accessibility Lab*, was developed. The application incorporated both accessibility-compliant features and deliberate accessibility violations aligned with the criteria in Table I. Examples included missing text alternatives, incorrect semantic roles (e.g., using a generic container instead of a button), insufficient colour contrast, and misaligned focus behaviour.

Each accessibility issue was explicitly mapped to a specific UI component or interaction, establishing a traceable link between the normative requirement and its test scenario. This mapping enabled a controlled comparison of tool detection capabilities while reducing evaluator subjectivity. The embedded issues functioned as known ground truths against which each tool's performance could be assessed.

C. Evaluation Procedure

Tool selection was informed by prior research [12]–[14] and official documentation on Android accessibility testing practices [15]. The chosen tools represent a range of evaluation methods, which can be categorised into automated methods (Lint and the Accessibility Test Framework), semi-automated methods (Google's Accessibility Scanner), and manual methods (the inbuilt Android screen-reader).

- **Lint (Static Code Analysis):** Source code was scanned using Android Studio's Lint framework to identify issues such as missing content labels, duplicate view IDs, and layout warnings.
- **Accessibility Test Framework (ATF):** JUnit-based test suites were executed using the Accessibility Test Framework

TABLE I. Condensed criteria for native Android applications derived from EN 301 549 (Section 11 - Software).

Category	Ref.	Generalised Requirement
Text alternatives	11.1.1.x	Non-text content must provide meaningful programmatic alternatives.
Time-based media & audio control	11.1.2.x, 11.1.4.2	Audio and video content must provide accessible alternatives and controllable playback.
Info & relationships	11.1.3.1	Content must expose a consistent semantic hierarchy.
Use of colour and contrast	11.1.4.1, 11.1.4.3	Information must not rely on colour and must meet contrast requirements.
Resizable text	11.1.4.4	Text must remain usable when scaled without loss of functionality.
Navigation & focus	11.2.1.1, 11.2.4.3	Interfaces must support keyboard navigation and provide a logical focus order.
Input assistance	11.3.3.x	Form fields must be properly labelled; errors must be identifiable and correction support provided.
Status messages	11.4.1.3	Changes in status must be programmatically detectable.
Accessibility services	11.5.2.x	Applications must interoperate with assistive technologies and avoid gesture-exclusive interactions.

to validate semantic roles, focus order, and error messaging in runtime behaviour.

- **Accessibility Scanner (Interactive Scanning):** Google's on-device evaluation tool was applied to runtime screens to detect visual issues, such as insufficient colour contrast and inadequate touch target sizes.
- **TalkBack (Manual Screen Reader Testing):** The application was manually tested with TalkBack, Android's built-in screen reader, to assess real-time spoken feedback, focus navigation, and usability from a user perspective.

Each tool was applied independently to the same version of the *Accessibility Lab* application. For every predefined accessibility issue, it was recorded whether the tool correctly identified the problem. Alongside these detection outcomes, observations were made about how each tool integrated into development workflows and how usable it was in practice. This information informed the later comparison of tool effectiveness for use in realistic Android development contexts.

IV. Results

The evaluation examined how effectively each selected tool detected predefined accessibility issues in the prototype application. Overall, the tools exhibited complementary strengths across structural (code-level), visual (presentation), and experiential (user interaction) aspects of accessibility.

Table II summarizes each tool's performance across four aspects: standards coverage, accuracy, integration potential, and developer usability.

TABLE II. Comparison of evaluation tools and their performance.

Evaluation Criterion	Lint	ATF	Scanner	TalkBack
Standards coverage	medium	high	medium	high
Accuracy	high	high	medium	medium
IDE/CI integration	high	high	low	low
Usability (for devs)	medium	medium	high	medium

Rating scale: **high** = reliable or fully met; **medium** = partly met; **low** = barely or not met.

Standards coverage: ATF and TalkBack were the only tools rated "high". Lint and the Accessibility Scanner, by contrast, offered limited depth and missed critical runtime issues.

Accuracy: Lint and ATF demonstrated low false positive rates, meaning that most of the flagged issues were genuine accessibility violations. The Accessibility Scanner and TalkBack, in comparison, required more interpretive/subjective judgement, particularly for interactive elements and user feedback.

Integration potential: Lint and ATF offered higher development *integration potential*, supporting repeatable checks within the Integrated Development Environment (IDE) and Continuous Integration (CI) pipelines, whereas the Accessibility Scanner and TalkBack operated independently of IDEs and did not support automation.

Developer usability: The Accessibility Scanner was the only tool rated "high", because it runs directly on the device with a simple UI and allows for easy tap and scan operations to immediately see issues on the screen. The other tools were rated "medium" - Lint messages can be somewhat technical and require accessibility knowledge in order to understand and act on them, which the average developer might not possess. Although the ATF is very powerful, it requires the writing of JUnit tests and the ongoing maintenance of test code, and TalkBack requires the user to first learn screen-reader gestures.

Table III presents a tool coverage matrix for five aggregated accessibility criteria. These criteria were derived from the categories in Table I, grouping related requirements into broader themes. Both ATF and TalkBack demonstrated full coverage across all evaluated categories. Lint primarily addressed structural issues, such as missing labels, while the Accessibility Scanner detected visual design flaws but did not assess interaction behaviour.

TABLE III. Coverage of key accessibility criteria by each tool.

Accessibility Criterion	Lint	ATF	Scanner	TalkBack
UI recognizability (e.g., Ref. 11.1.1.x)	✓	✓	✗	✓
Semantic roles (e.g., Ref. 11.1.3.1)	✗	✓	✓	✓
Navigation and focus (e.g., Ref. 11.2.1.1)	✗	✓	✗	✓
Error identification (e.g., Ref. 11.3.3.x)	✗	✓	✗	✓
Alt input methods (e.g., Ref. 11.5.2.x)	✗	✓	✗	✓

Rating scale: ✓ = criterion addressed; ✗ = not addressed.

Taken together, these results indicate that none of the evaluated tools combined consistently strong performance across all evaluation aspects with comprehensive coverage of the assessed EN 301 549 accessibility criteria. Instead, each method addressed distinct subsets of structural, visual, and experiential issues, which reinforces the need for a multi-method evaluation strategy.

Static analysis via Android Lint effectively identified structural violations, such as missing labels and semantic roles. Its integration with IDE workflows and CI pipelines supports early-stage and repeatable code validation. However, Lint did not detect runtime or experiential issues, including navigation flow or user feedback.

The ATF extended code analysis to include semantic assertions and interaction-level checks. It aligned well with the selected EN 301 549 criteria, and it supported automation through JUnit integration, which makes it suitable for regression testing. Nevertheless, ATF was unable to assess perceptual and context-sensitive aspects, such as the clarity of spoken feedback or how updates are experienced by screen reader users.

Manual screen reader testing using TalkBack addressed these limitations by revealing dynamic issues, such as focus order, announcement of state changes, and overall usability from the perspective of screen reader users. TalkBack matched the coverage of ATF on the evaluated criteria and additionally exposed experiential problems that automated techniques failed to capture. However, this approach requires expertise in assistive

technology and could not be integrated into automated pipelines, limiting its effectiveness in software development projects.

The Accessibility Scanner proved highly usable for on-device inspections. It effectively flagged visual design flaws, such as insufficient contrast and small touch targets, which is consistent with its focus on layout and presentation. At the same time, it provided limited insight into semantic and behavioural issues and did not support automation.

Overall, the tools exhibited complementary strengths. Automated tools such as Lint and the ATF enable scalable code validation within development workflows, while semi-automated visual inspection methods like the Accessibility Scanner and manual methods such as the TalkBack screen reader are essential for identifying user experience and behaviour-related issues. A coordinated, multi-method approach is therefore necessary to achieve robust accessibility evaluation that remains consistent with the assessed EN 301 549 accessibility requirements and is practical for use in contemporary Android development.

V. Limitations

Several limitations should be considered when interpreting the findings. First, the evaluation used a single prototype application - *Accessibility Lab*. Although it included typical accessibility issues, it does not reflect the full diversity of real-world Android apps. Second, only freely available testing tools were examined. Commercial solutions, which may provide additional capabilities, were not considered. Third, the qualitative ratings in Table II, particularly for developer usability, involved subjective judgement. Independent replication by 3rd-party experts would strengthen the reliability of this rating.

VI. Conclusion and Future Work

This study evaluated four freely available tools for assessing the accessibility of native Android applications against the EN 301 549 requirements. By applying a structured evaluation strategy to a prototype application, the analysis revealed distinct strengths and limitations across automated (Lint and the ATF), semi-automated (the Accessibility Scanner), and manual (the TalkBack screen reader) accessibility inspection methods.

The study confirmed that no single tool provided both consistently strong performance across all evaluation aspects and broad coverage of the assessed accessibility criteria, but that a combined-methods approach enables comprehensive and standards-aligned evaluation.

A key contribution was the structured guidance showing the effective integration of accessibility tools into Android development workflows, and because all tools are freely available and require minimal infrastructure, the proposed multi-method strategy is practical for contemporary Android development environments.

Multiple avenues for research and tool advancement could follow from this study. The evaluation could be extended to production-scale applications to provide real-world validation, the *Accessibility Lab* testbed application could be made available for others, and the evaluation could also incorporate user-centred assessment by individuals with disabilities.

References

[1] Statcounter Global Stats, "Mobile Operating System Market Share Europe", Web statistics, 2025, Accessed: Mar. 12, 2026. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/europe

[2] World Health Organization, Regional Office for Europe, "Vision and hearing loss", Fact sheet, 2023, Accessed: Mar. 12, 2026. [Online]. Available: https://www.who.int/europe/news-room/fact-sheets/item/vision-and-hearing-loss

[3] M. Di Gregorio, D. Di Nucci, F. Palomba, and G. Vitiello, "The making of accessible Android applications: an empirical study on the state of the practice", *Empirical Software Engineering*, vol. 27, no. 6, p. 145, 2022. DOI: 10.1007/s10664-022-10182-x

[4] *Directive (EU) 2016/2102 of the European Parliament and of the Council of 26 October 2016 on the accessibility of the websites and mobile applications of public sector bodies*, Official Journal of the European Union, L 327, OJ L 327, 2.12.2016, p. 1–15, Dec. 2016.

[5] *Directive (EU) 2019/882 of the European Parliament and of the Council of 17 April 2019 on the accessibility requirements for products and services*, Official Journal of the European Union, L 151, OJ L 151, 7.06.2019, p. 1–46, Jun. 2019.

[6] *EN 301 549 V4.1.0 (2025-11): Accessibility Requirements for ICT Products and Services*, Version 4.1.0, November 2025, Sophia Antipolis, France: European Telecommunications Standards Institute (ETSI), 2025.

[7] J. Noraman, "Anforderungen an die Barrierefreiheit in mobilen Anwendungen [Accessibility Requirements for Mobile Applications]", de, M.S. thesis, Fakultät III - Medien, Information und Design, 2022, p. 54. DOI: 10.25968/opus-2370

[8] L. Seixas Pereira, M. Matos, and C. Duarte, "Exploring Mobile Device Accessibility: Challenges, Insights, and Recommendations for Evaluation Methodologies", in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, NY, USA: ACM, 2024, pp. 1–17. DOI: 10.1145/3613904.3642526

[9] M. Olsson and V. Zubenko, "Breaking Barriers: The Challenges of Accessibility in Mobile Application Development", M.S. thesis, 2024.

[10] A. Alshayban and S. Malek, "AccessiText: automated detection of text accessibility issues in Android apps", in *Proceedings of the 30th ACM Joint European Software Engineering Conference on the Foundations of Software Engineering*, NY, USA: ACM, 2022, pp. 984–995. DOI: 10.1145/3540250.3549118

[11] A. Indika et al., "Exploring Accessibility Trends and Challenges in Mobile App Development: A Study of Stack Overflow Questions", in *Proceedings of the 58th Hawaii International Conference on System Sciences*, 2025, pp. 7401–7410. DOI: 10.24251/HICSS.2025.885

[12] P. Gersbacher, "Analyse von automatisierten Testverfahren für die Barrierefreiheitsprüfung von Apps im Hinblick auf den Europäischen Standard EN 301 549 [Analysis of Automated Testing Methods for Assessing the Accessibility of Apps in Accordance with EN 301 549]", de, M.S. thesis, 2021.

[13] E. Panula, "Towards More Accessible Android Applications: An Actionable Accessibility Checklist for Android Developers", M.S. thesis, 2024.

[14] M. Shoaib, D. Fitzpatrick, and I. Pitt, "Accessibility Features of Developmental Platforms: Towards Developing Accessible Mobile Applications with Cross-platform, Research Challenges and Opportunities", in *Proceedings of the 5th International Workshop on Digitization and E-Inclusion in Mathematics and Science*, 2024, pp. 153–162.

[15] Google, "Test your app's accessibility", Android Developers documentation, 2026, Accessed: Mar. 12, 2026. [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility/testing