



UNIVERSITY OF PADUA  
UNIVERSITA' DEGLI STUDI DI PADOVA

---

# Designing an accessibility learning toolkit - Bridging the gap between guidelines and implementation

---

Department of Mathematics "Tullio Levi-Civita"  
Master Degree in Computer Science

---

**Graduate:** Gabriel Rovesti  
**Supervisor:** Prof. Ombretta Gaggi

Graduation Session: 25/07/2025

**Definition:** Ability for users to fully perceive, understand, navigate, and interact with digital content regardless of capabilities

## The mobile reality:

- 1.3 billion people worldwide live with disabilities (WHO, 2023)
- **Mobile-first** era: 6.8 billion smartphone users globally
- Mobile interfaces create new **accessibility barriers** for assistive technology users



## Touch interaction barriers:

- **Target size** not standardized and difficult to use
- **Complex gestures** might exclude different categories of users
- **One-handed operation** limitations

## Mobile context issues:

- **Small screens** affect content hierarchy
- **Orientation changes** disrupt navigation
- **Performance impact** on battery and processing



## Current standards:

- **WCAG 2.2 (2023):** 4 principles, 3 levels of conformance - **web-focused**
- **MCAG (2015):** Mobile adaptation - **based on WCAG 2.0**
- **WCAG2Mobile (2025):** Recent mobile guidance - **interpretations only**



## The problem:

- **Outdated foundation:** MCAG missing WCAG 2.1/2.2 mobile criteria
- **Implementation void:** No practical framework for mobile developers
- **Knowledge fragmentation:** Scattered resources, unclear costs



**Result:** Mobile developers lack comprehensive accessibility implementation guidance

## Platform differences:

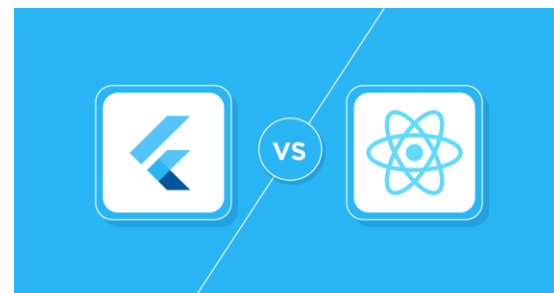
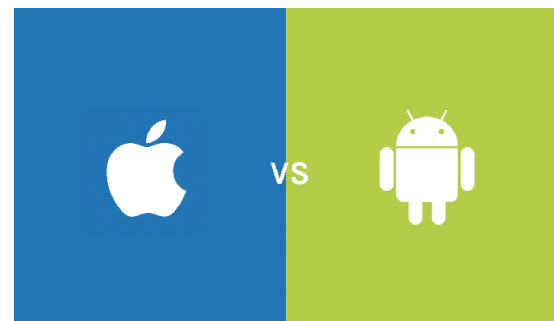
- **iOS:** VoiceOver, Voice Control, Switch Control
- **Android:** TalkBack, Switch Access, Keyboard navigation

## Framework responses:

- **Flutter:** Single accessibility tree, platform adaptation layer
- **React Native:** Platform-specific accessibility props, native bridge

## The developer challenge:

- **Budai (2024):** Tested component accessibility → fragmented knowledge
- **Gap identified:** No comprehensive learning resource bridging platforms



## Research Questions as standard approach:

- **RQ1:** Are React Native components accessible by default?
- **RQ2:** Can non-accessible components be made accessible?
- **RQ3:** What's the implementation cost (code overhead)?

**AccessibleHub:** React Native application **tested on both Android and iOS** serving as **interactive accessibility manual** for mobile developers






- **Every screen analyzed** for accessibility patterns and costs
- **Educational platform** bridging theory to practice



A comprehensive toolkit for implementing accessibility in React Native

React Native v0.73 WCAG 2.2 Expo SDK

## Core sections:

-  **Accessible Components:** UI implementations with copyable code
-  **Best Practices:** Educational content on accessibility challenges
-  **Tools - Settings:** Resource catalog for testing and common settings
-  **Framework Comparison:** Evidence-based evaluation methodology
-  **Instruction & Community:** Social learning & collaborative resources

**Research innovation:** Every screen analyzed as case study

- **Dual methodology:** Both research vehicle & educational tool
- **20+ components tested** with TalkBack and VoiceOver
- **Cross-platform validation** ensuring patterns work universally



**The transformation challenge:** WCAG guidelines are abstract and difficult to implement directly in mobile code

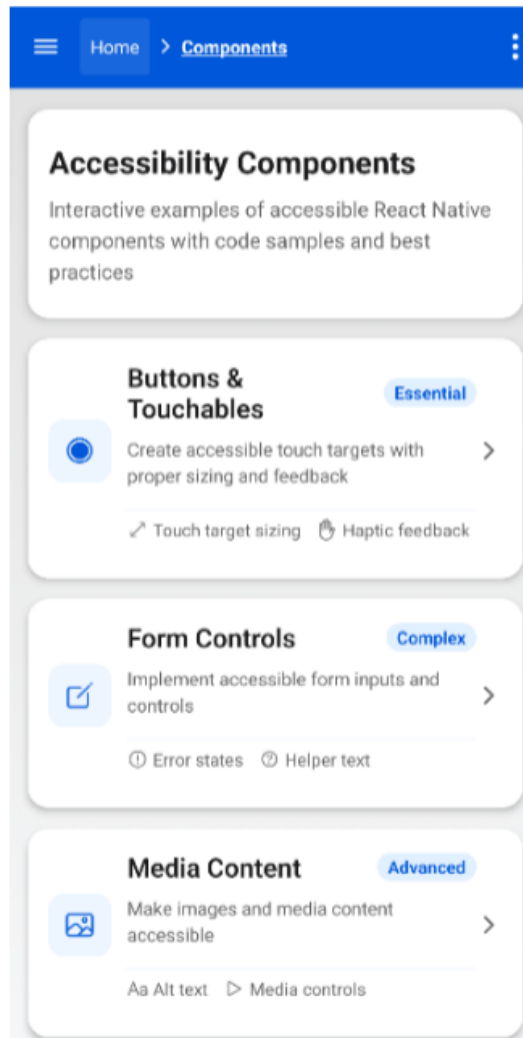
Layer	Input	Output	Example
<b>Theoretical Foundation</b>	WCAG abstract principles	Success criteria	"Content must be perceivable"
<b>Implementation Patterns</b>	Success criteria	React Native code	<code>accessibilityLabel="Save document"</code>
<b>Screen-Based Analysis</b>	Code patterns	Quantified metrics	13.3% overhead for buttons

**Basic workflow** - Enabling data-driven accessibility decisions

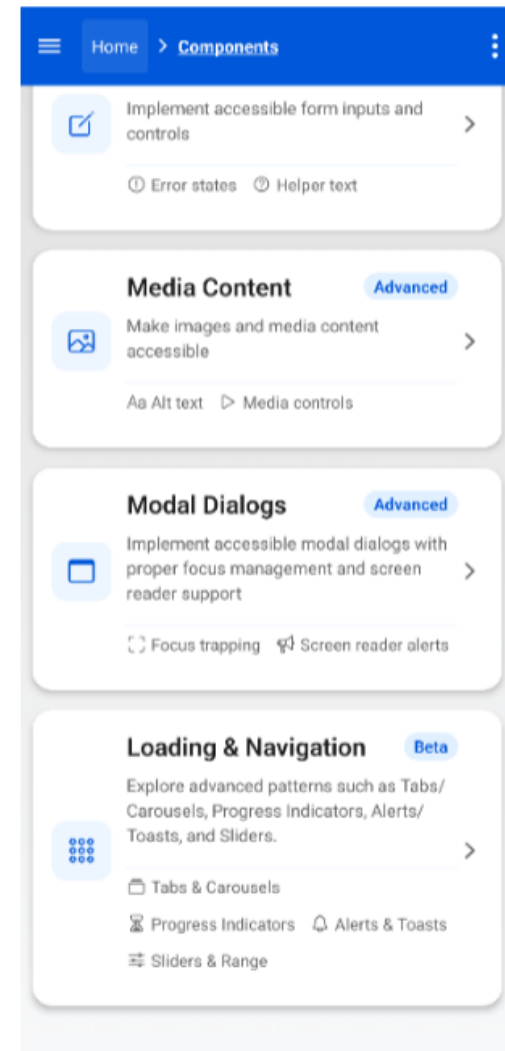
Abstract WCAG → Implementation Patterns →  
Quantified Metrics → Educational Platform



# Systematic analysis – Example (1)



(a) Components screen - Top section



(b) Components screen - Bottom section

# Systematic analysis – Example (2)



**Table 3.5:** Components screen component-criteria mapping with WCAG2Mobile considerations

Component	Semantic Role	WCAG 2.2 Criteria	WCAG2Mobile Considerations	Implementation Properties
ScrollView Container (main screen container)	scrollview	2.1.1 Keyboard (A) 2.4.3 Focus Order (A)	Screen-based navigation patterns; Touch-based scrolling alternatives	accessibility Role="scrollview", accessibility Label="Accessibility Components Screen"

**Table 3.6:** Components screen screen reader testing with WCAG2Mobile focus

Test Case	VoiceOver (iOS)	TalkBack (Android)	WCAG2Mobile Considerations
Screen Title	✓ Announces “Accessibility Components, heading”	✓ Announces “Accessibility Components, heading”	SC 1.3.1 and 2.4.6 interpreted for screens instead of web pages

**Table 3.7:** Components screen accessibility implementation overhead

Accessibility Feature	Lines of Code	Percentage of Total Code	Complexity Impact
Semantic Roles	15 LOC	2.6%	Low
Descriptive Labels	28 LOC	4.9%	Medium

**Key finding:** Accessibility implementation requires 12-23% additional code across component types

Component Type	Complexity Level	Code Overhead	Primary Contributors
Media	Low	12.7%	Alt text, captions
Buttons	Low	13.3%	Semantic roles, labels
Dialogs	Medium	16.2%	Focus management
Forms	Medium	21.5%	State management, error handling
Advanced	High	22.7%	Custom controls, gestures

## Critical insights:

- **Even complex components stay under 25%**
- **Correlation** between interaction complexity and implementation cost
- **Manageable overhead** for significant usability improvements
- **First quantitative framework** for mobile accessibility cost assessment

**Innovation:** Evidence-based methodology for quantifying mobile accessibility implementation across frameworks

 **Implementation Overhead (IMO)**  
Direct code cost measurement for equivalent functionality

 **Screen Reader Support Score (SRSS):**  
Likert scale based on VoiceOver/TalkBack functionality

 **WCAG Compliance Ratio (WCR):**  
Standards adherence tracking (A/AA/AAA levels)

 **Complexity Impact Factor (CIF):**  
Development difficulty classification (Low/Medium/High)

 **Development Time Estimate (DTE):**  
Resource planning with complexity adjustments



## Architecture differences:

- **React Native:** Property-based model (accessibilityLabel, accessibilityRole)
- **Flutter:** Widget-based approach (explicit Semantics wrappers)

Metric	React Native	Flutter	Decision Factor
Implementation Overhead	45% less code	Baseline	Development speed
Screen Reader Support	4.2/5.0	3.8/5.0	User experience
Default Accessibility	38%	32%	Both require intervention
Architecture	Property-based	Widget-based	Code complexity
Learning Curve	Moderate	Steep	Team onboarding

**REACT NATIVE**



**vs**



**FLUTTER**

# Framework comparison results (1)



Component	React Native	Flutter	Code Overhead	Screen Reader Support
Text Language	✓ Default	✗ Manual	Flutter +200%	RN: 4.2, FL: 3.7
Headings	✗ Manual	✗ Manual	Flutter +57%	RN: 4.3, FL: 4.0
Form Fields	✗ Manual	✗ Manual	Flutter +53%	RN: 4.0, FL: 3.8
Custom Gestures	✗ Manual	✗ Manual	Flutter +27%	RN: 3.8, FL: 3.2
OVERALL	38% Default	32% Default	Flutter +119%	RN: 4.2, FL: 3.8

## Key Patterns Identified:

- **Text language declaration:** Largest overhead difference (Flutter +200%)
- **Custom gestures:** Smallest gap (Flutter +27%) - both frameworks struggle
- **Default accessibility:** React Native provides more out-of-box features (38% vs 32%)
- **Screen reader consistency:** React Native scores higher across all component types

# Framework comparison results (2)



Metric	React Native	Flutter	Key Finding
Default Accessibility	38%	32%	RN +6% advantage
Implementation Overhead	Baseline	+119% more code	RN significantly more efficient
Screen Reader Support	4.2/5	3.8/5	RN better cross-platform consistency
WCAG Compliance (AA)	92%	85%	RN +8.2% higher compliance

## Critical insights:

- Choose **React Native** when: Rapid development, web accessibility experience, tight deadlines
- Choose **Flutter** when: Complex custom components, long-term maintenance teams, granular control

## Key contributions:

- Extended research framework from Flutter-only to comparative analysis
- First quantitative framework for mobile accessibility cost assessment
- 45% implementation overhead reduction with React Native
- Systematic methodology bridging WCAG theory to mobile practice

## Research answers:

- **RQ1:** Neither framework accessible by default (38% vs 32%)
- **RQ2:** Both achieve 100% WCAG compliance with proper implementation
- **RQ3:** React Native requires 45% less code for equivalent accessibility

