

Elevated Testing Depth and Quality for Design Implementations through Screenshot Testing

Thesis defense

Alexandru-Vlad Vamoş

Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

25 March 2025

Supervisors

Christian Schrödel
Prof. Dr. Dirk Riehle, M.B.A.

Introduction

■ Design implementation

- Large amounts of effort
- **Manual verification** remains the most common
 - Dependency on human effort
 - Poor scalability

■ Goal(s)

- 1 **Strategy** for automating design verification in Android applications using **screenshot testing**: "*[An] effective way to verify how your UI looks to users.*" ("Compose Preview Screenshot Testing", 2024).
- 2 **Road map** for teams that seek to implement their own testing strategy

■ Challenges

- Choosing an adequate testing tool
- Formulating what components of the application should be tested
- Integration into the CI/CD pipeline

Test Automation

Relevant testware from Baumgartner et al. (2022):

- Software
- Test data
- Test environment

Central objectives acc. to Baumgartner et al. (2022):

- "improve test efficiency and thus reduce the overall cost of testing" (p. 9),
- "[maintain] or [increase] quality" (p. 10).

Snapshot Testing

- Lack of academic work on screenshot testing, unlike snapshot testing
- Snapshot testing:
 - "a type of output comparison testing technique that asserts whether the outputs by the current state of the product remain unchanged" (Fujita et al., 2023, p. 335)
 - verifies whether the implementation "matches a certain golden standard" (Cruz et al., 2023)
 - outputs: DOM trees, JSON objects, PNGs...
- Screenshot testing is a category of snapshot testing

Cruz et al. (2023) (gray literature review):

- Benefits:
 - "easy to implement" (p. 4)
 - "prevents regression" (p. 4)
- Drawbacks:
 - no alternative to end-to-end UI tests
 - snapshot size
 - fragility

Requirements

Functional requirements:

- Enables automated screenshot regression testing
- High-level API
- Create and store reference screenshots
- Local execution
- CI/CD integration
- Meaningful outputs

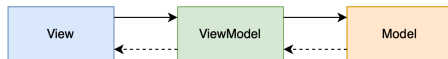
Quality goals:

- Functional suitability
- Efficiency and Performance
- Reliability
- Maintainability
- Scalability

Architecture

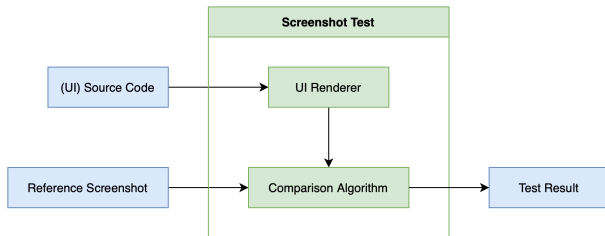
■ Application Architecture

- Screenshot testing targets the **view** component



■ Screenshot test structure

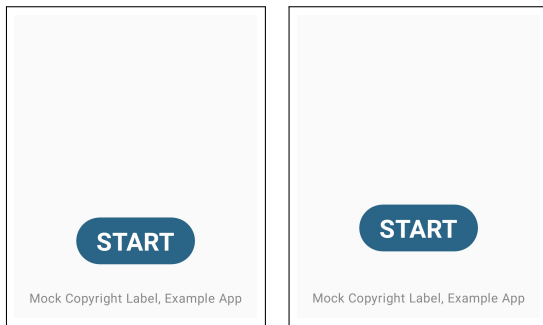
- Inputs: UI code, reference screenshots
- Output: test result



Comparison Algorithms

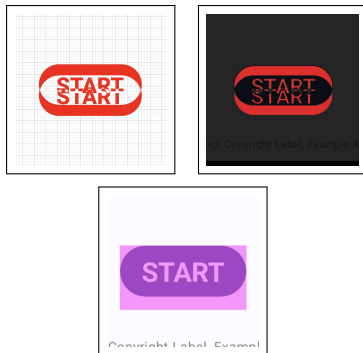
- **Open-source, pixel-by-pixel**
 - each pixel compared to its direct counterpart
- **Closed-source, allegedly enhanced**
 - hard to analyse
- Comparison using three representative UI regression scenarios
 - OS, PbP: Takahiro Menju's *Roborazzi*
 - CS, enhanced: *App Percy* from Browserstack, *Aplitools Eyes* from Aplitools

First Scenario



Original (left), Regression (right)

Comparison



Roborazzi (top left), App Percy (top right), Aplitools Eyes (bottom)

- Constant behaviour across scenarios
- Further adv. and disadv.
 - App Percy, Aplitools
 - advanced online platforms (local hosting?)
 - no contextual information or solution recommendation
 - Roborazzi
 - simplicity
 - community contributions
- No compelling reasons to recommend closed-source, allegedly enhanced, tools and algorithms over open-source tools that use conventional pixel-by-pixel comparison algorithms

Screenshot Testing Tool

Candidates (widely-adopted and under active development)

- Pedro Gomez's *Shot*
- Cashapp's *Paparazzi*
- Takahiro Menju's *Roborazzi*

Comparison (with insights from Soares (2023a)):

Criterion / Tool	Shot	Paparazzi	Roborazzi
Total Required Effort	minimal	minimal	minimal
Requires Instrumentation	yes	no	no
Execution Speed	slower	fast	fast
Support for UI Variations	hard	easy	easy
Supports UI Interaction	yes	no	yes

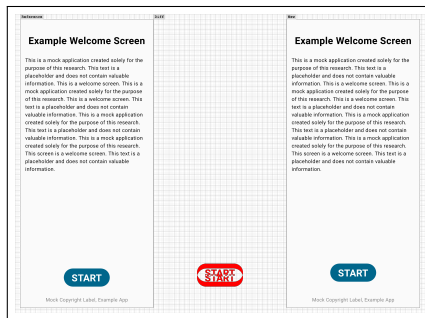
The overall best choice is Roborazzi.

Roborazzi Screenshot Test Example

```
1 @RunWith(androidx.test.ext.junit.runners.AndroidJUnit4::class)
2 @GraphicsMode(GraphicsMode.Mode.NATIVE)
3 @Config(qualifiers = RobolectricDeviceQualifiers.Pixel7Pro)
4 class HomeScreenTest {
5     private val customImageComparator = SimpleImageComparator(
6         maxDistance = 0.01f,
7         hShift = 2,
8         vShift = 2
9     )
10    private val customThresholdValidator = ThresholdValidator(0.02F)
11
12    @OptIn(ExperimentalRoborazziApi::class)
13    @get:Rule
14    val roborazziRule = RoborazziRule(
15        RoborazziRule.Options(
16            outputDirectoryPath = "baseline—screenshots",
17            roborazziOptions = RoborazziOptions(
18                compareOptions = RoborazziOptions.CompareOptions(
19                    imageComparator = customImageComparator,
20                    resultValidator = customThresholdValidator
21                )
22            )
23        )
24    )
25
26    @Test
27    fun testHomeScreen() {
28        captureRobolImage { AppTheme { HomeScreen() } }
29    }
30 }
```

Execution and Outputs

- Execution through Gradle tasks:
 - `recordRoborazziDebug` records reference screenshots
 - `verifyRoborazziDebug` runs the tests
- HTML reports with difference images:



Testing Strategy

1 What parts of the UI should be tested?

- not all test objects yield the same value
- test object granularity: low-level UI element \leq test object \leq screen
- define *design system*
 - "explains how a team should create products" (Perez-Cruz, 2019, p. 2)
 - includes a UI component library (Fanguy, 2019)

Recommendation based on Soares (2023b):

- Always implement **screen-level** screenshot tests
 - large feature coverage
 - end user perspective
- If a design system exists, screenshot test individual **low-level UI components** as well
 - early regression detection
 - easier bug localization

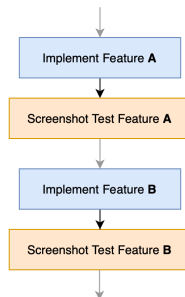
Testing Strategy

2 When should screenshot tests be implemented?

- added value can depend on timing

Recommendation:

- Screenshot tests should be implemented right after the UI feature implementation
 - raises test coverage
 - prevents test debt



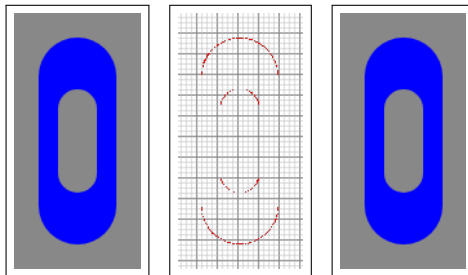
CI/CD Integration

- Widely-used DevOps practice
- Android: build → ... → test → ... → deliver
- Roborazzi:
 - all inputs in project
 - execution through Gradle tasks
 - only a simple Gradle task execution is required:

```
1 stage('Run Roborazzi Screenshot Tests') {  
2     sh('./gradlew verifyRoborazziDebug')  
3 }
```

Hidden Benefit

- Rendering differences across OSs
 - "due to variations in how graphics libraries render components on different platforms" ("Roborazzi FAQ", 2024)



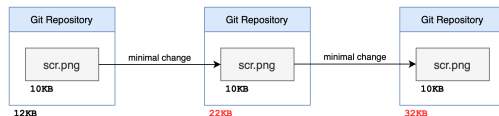
MacOS render (left), Diff (center), Windows render (right)

- recording and testing in the pipeline keeps differences constant

Data Management

1 Git (with no extra setup)

- "Vanilla"
- Benefits
 - no extra setup
 - simplicity
- Drawback
 - Git's problematic handling if non-text binary files (such as PNGs):



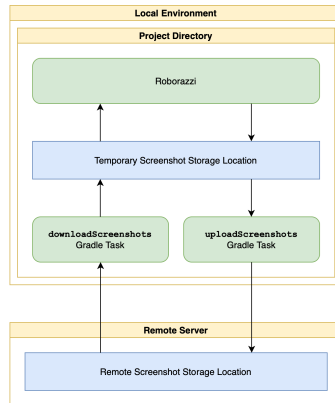
2 Git Large File Storage (LFS)

- Open-source Git extension
- Files in repository are replaced by pointers
- Improves the experience when handling large binary files
- Benefits
 - simple setup
 - seamless integration into Git workflow

Data Management

3 Custom Storage and Versioning Solution

- Remote storage
- Data transfer through Gradle tasks
- Benefit
 - high flexibility
- Drawbacks
 - high effort
 - digital infrastructure



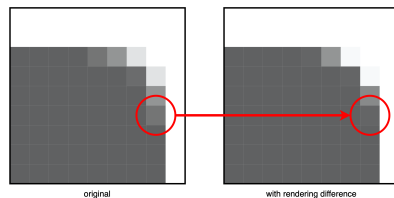
Recommendation: data management solution that accounts for Git's shortcomings

Pixel Differences and Tolerance Parameters

Pixel differences can generally have two causes:

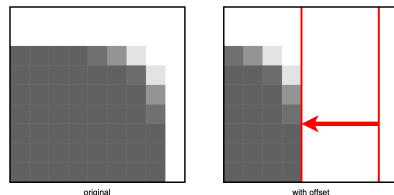
1 Platform-dependent rendering differences

- Different implementations of graphics libraries (Menju, 2024)
- Tolerance parameters:
 - maxDistance
 - vShift, hShift



2 UI Regressions

- This is what screenshot tests aim to detect
- Tolerance parameter:
 - threshold

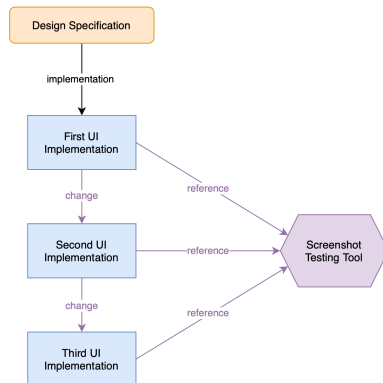


Project Demo

- Demo Android application
- Roborazzi
- Testcase implementation strategy
- Git LFS
- Integration into CI/CD pipeline

Evaluation









- Quality goals have been *achieved* (Functional Suitability, Efficiency and Performance, Reliability, Maintainability, Scalability)
- Main shortcoming: fails to automate the **complete** design verification process (see →)
- Complete automation is a possible topic for future research



Conclusion

- Successfully created a screenshot testing strategy
- Topics tackled
 - types of available tools
 - tool comparison
 - CI/CD integration
 - testcase implementation strategy
 - platform-dependent rendering behaviour
- Topic by no means exhausted
 - future research: complete automation

References I

-  Baumgartner, M., Steirer, T., Wendland, M.-F., Gwihs, S., Hartner, J., & Seidl, R. (2022). *Test automation fundamentals*. Rocky Nook.
-  Compose preview screenshot testing [Accessed: 31.10.2024]. (2024).
<https://developer.android.com/studio/preview/compose-screenshot-testing>
-  Cruz, V. P. G., Rocha, H., & Valente, M. T. (2023). Snapshot testing in practice: Benefits and drawbacks. *The Journal of Systems and Software*, 204.
-  Fanguy, W. (2019). *A comprehensive guide to design systems*. Retrieved October 30, 2024, from
<https://www.invisionapp.com/inside-design/guide-to-design-systems/>
-  Fujita, S., Kashiwa, Y., Lin, B., & Iida, H. (2023). An empirical study on the use of snapshot testing. *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 335–340.
-  Menju, T. (2024). Comment on issue #351 in the *Roborazzi* repository [Accessed: 31.10.2024]. <https://github.com/takahirom/roborazzi/issues/351#issuecomment-2100757751>
-  Perez-Cruz, Y. (2019). *Expressive design systems*. A Book Apart.
-  Roborazzi faq [Accessed: 30.10.2024]. (2024).
<https://takahirom.github.io/roborazzi/faq.html>

References II



Soares, U. (2023a). *The landscape of android screenshot testing in 2023*. Retrieved October 27, 2024, from <https://ubiratansoares.dev/posts/screenshot-testing-for-android-landscape/>



Soares, U. (2023b). *Two strategies to drive screenshot testing in mobile projects*. Retrieved October 27, 2024, from <https://ubiratansoares.dev/posts/two-strategies-for-screenshot-testing/>

Thank you for your attention!

Questions? Feedback?