Identifying Architectural Technical Debt in Android Applications through Automated Compliance Checking

Roberto Verdecchia
roberto.verdecchia@gssi.it
Architectural Technical Debt (ATD)

- **Sub-optimal decisions** resulting in **immature architectural artifacts**
- Hinders **maintainability** and **evolvability**
- **ATD identification**: detecting ATD during or after architecting processes
- To date, few approaches consider Android specific ATD

---

Research question

“How can we identify automatically Architectural Technical Debt specific to Android applications?”
Approach overview
Approach overview

1. Extraction of Android architectural guidelines
Approach overview

1. Extraction of Android architectural guidelines
2. Establishment of Android reference architecture
Approach overview

1. Extraction of Android architectural guidelines
2. Establishment of Android reference architecture
3. Reverse engineering of implemented architecture
Approach overview

1. Extraction of Android architectural guidelines
2. Establishment of Android reference architecture
3. Reverse engineering of implemented architecture
4. Compliance checking
Approach overview

1. Extraction of Android architectural guidelines
2. Establishment of Android reference architecture
3. Reverse engineering of implemented architecture
4. Compliance checking
5. Quantitative assessment of compliance violations
Identifying Architectural Technical Debt in Android Applications through Automated Compliance Checking

Overview
Mobile application business model is tightly coupled to users satisfaction. Promptly and efficiently releasing new versions to introduce new features, fix bugs, and adapt to users’ needs is crucial. Architectural technical debt (ATD) hinders by definition evolvability. Among other techniques, compliance checking is used to undercover ATD [1]. Recently, effort was spent in standardizing Android architectural components to lower complexity of Android apps and provide a Android architectural guidelines [2]. We present a novel approach, based on (i) guideline extraction, (ii) architecture reverse engineering and (iii) compliance checking, for identifying ATD hotspots in Android apps.

1. Guidelines extraction
The first step consists in an architectural guideline extraction from heterogeneous sources. Guidelines should embed architectural rules designed to avoid incurring in potential ATD. The extracted guidelines are validated through developers interviews.

4. Compliance checking
During this process, items of non-adherence of the implemented architecture w.r.t. the Android reference architecture are identified. The items are stored for a subsequent analysis carried out in Step 5. Due to its complexity this step has to be carried out (semi-) automatically through a model companion tool.

Example of guideline violation

5. Quantitative assessment
Once the set of non-adherence items is computed, it is possible to analyze the gathered data to identify which architectural elements of the implemented architecture violate the highest number of Android architectural guidelines. Distinct violation types can even be associated to specific weights to support a more involved prioritization process. The final architectural elements identified through this process are prioritized and corrected.

IN FEW WORDS
what (project goals):
- Reusable Android Reference Architecture
- Automated process for architectural technical debt hotspot analysis of Android applications

how (technique):
- Manual extraction and formalization of Android architectural guidelines
- Automated implemented architecture reverse engineering
- Automated model-based compliance checking
- Quantitative assessment of compliance violations to identify hotspot components

See you at the poster!
Identifying Architectural Technical Debt in Android Applications through Automated Compliance Checking

Roberto Verdecchia
roberto.verdecchia@gssi.it