Dynamic Design Analysis

Increasingly, software systems are composed at runtime. Yet, the impact of runtime composition on design quality is unknown. Static analysis, a state-of-the-practice approach, has demonstrated that dependency-caused design hotspots make security vulnerabilities more likely, but it does not detect the effect of dynamic dependencies.

In this work, we are creating tooling to identify dynamic dependencies as well as other information that is not available via static, parsing-based approaches, to both determine and augment the information that is missing in static approaches.

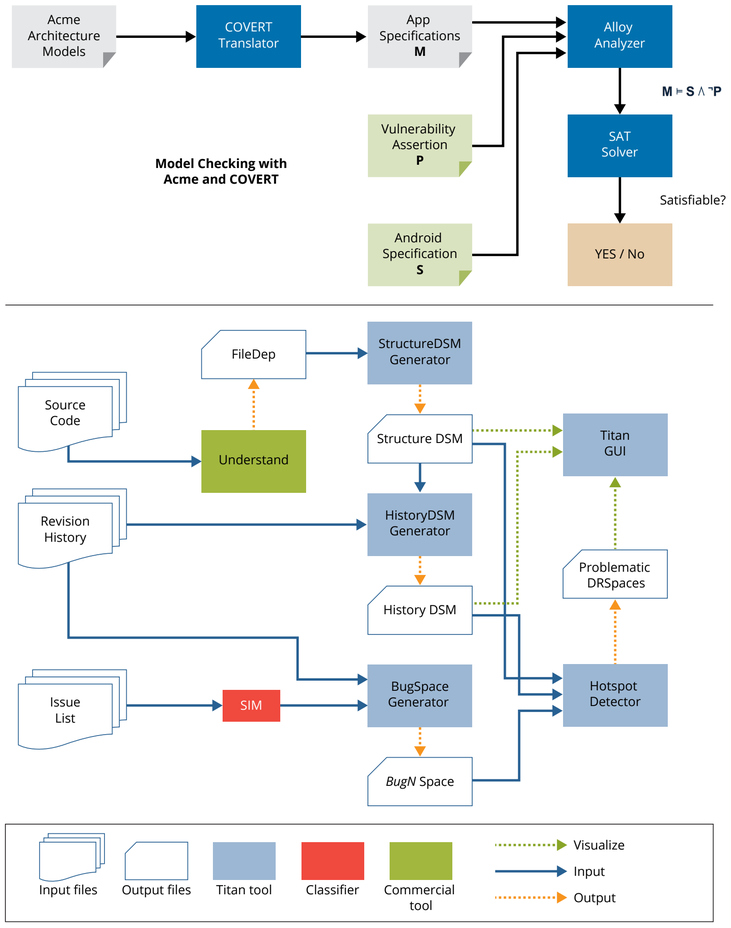
Technical challenges:

1. Detecting dynamic dependencies (DDs)
2. Determining whether DDs create new kinds of architectural flaws
3. Determining the consequences of DD-induced design flaws
4. Proposing refactorings to remedy flaws

Two approaches employed:

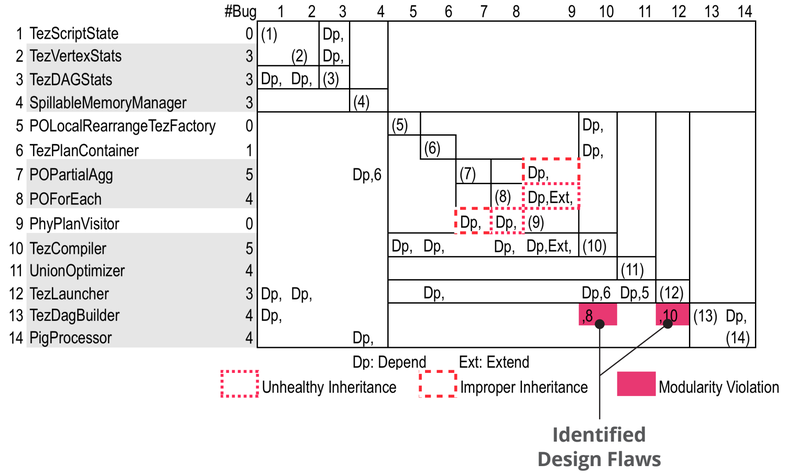
Approach 1: Acme/Alloy-based Analysis

- defined a formal architectural model of the Android framework as an architectural style in Acme
- used Soot to capture an abstract model of each app and translated these models to a single architecture in Acme
- defined Acme invariants to check architecture properties (e.g., apps cannot communicate implicit intents to apps that have a lower trust level)
- translated these models to Alloy specifications
  - checked that privilege escalation is not possible (or provide a counter-example where it is possible)



Approach 2: Titan-based analysis

- reverse-engineering of architecture facts using Understand
- built static representations of architecture dependencies as DSMs
- added historic dependencies from the revision history
- captured potential dynamic relationships as issues from the issue list
- defined a new architecture view: the issue space
- showed that when a system has files revised in many different issues—what we call a hotspot--these "shared" files are connected
- these design dependencies are frequently only seen at run-time and they almost always have design flaws leading to bugs, security flaws, and maintenance problems
- thus they should be analyzed and refactored



Conclusions and Future Work

Design flaws are introduced unknowingly by the daily activities of developers—adding features and fixing bugs. If left untreated they degrade the system over time, making it harder to understand, maintain, extend, and fix.

By considering dynamic information in conjunction with static information we can precisely locate such design flaws, and hence determine the root causes of bugs more quickly.

This information is not available solely via static analysis; dynamic dependencies must be considered. But such information is difficult to obtain. We have explored two methods for doing so.

These design flaws are the roots of technical debt.

In our future work, we are examining the relationship between such design flaws and security bugs.

Rick Kazman | rkazman@sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM17-0737
Dynamic Design Analysis