Research Review 2017

Dynamic Design Analysis

Rick Kazman

Dynamic Design Analysis

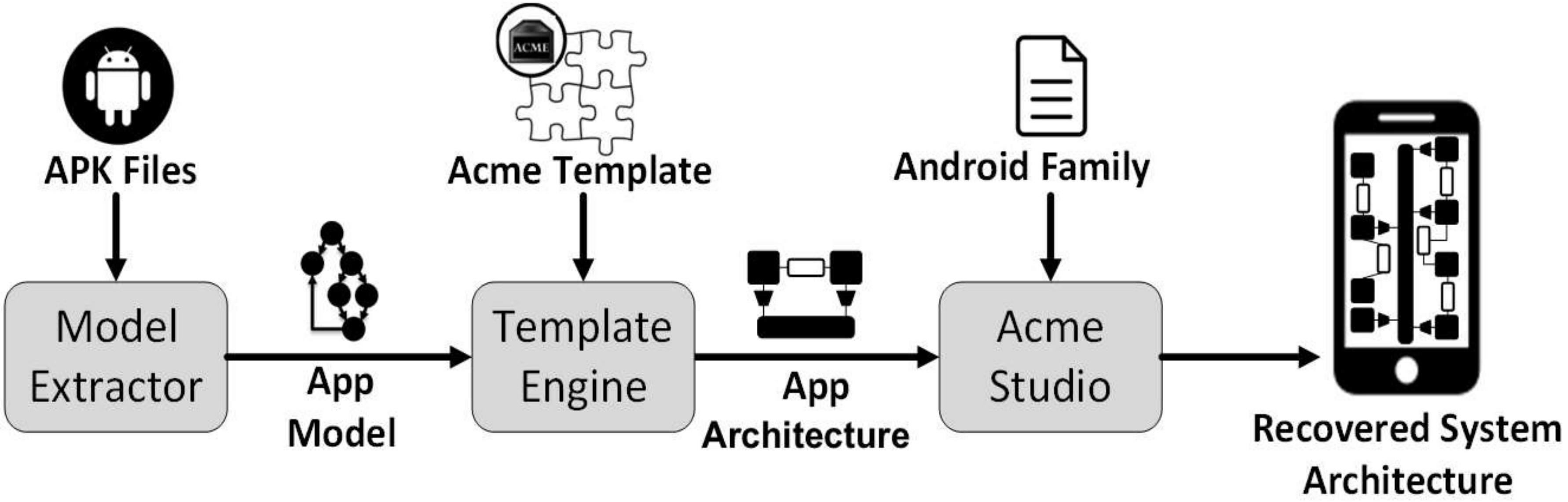
Problem: Increasingly, software systems are composed at runtime. Yet, the impact of runtime composition on design quality is unknown. Static analysis has shown that design flaws make bugs and security vulnerabilities more likely, but does not detect the effect of dynamic dependencies.

Solution: A technique to detect such flaws, either fully automatically (where possible) or semi-automatically (where necessary).

Main 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 these flaws.

Thrust 1: Recovering Android App Interactions



Thrust 1: Recovering Android App Interactions

Conformance Analysis Using ACME:

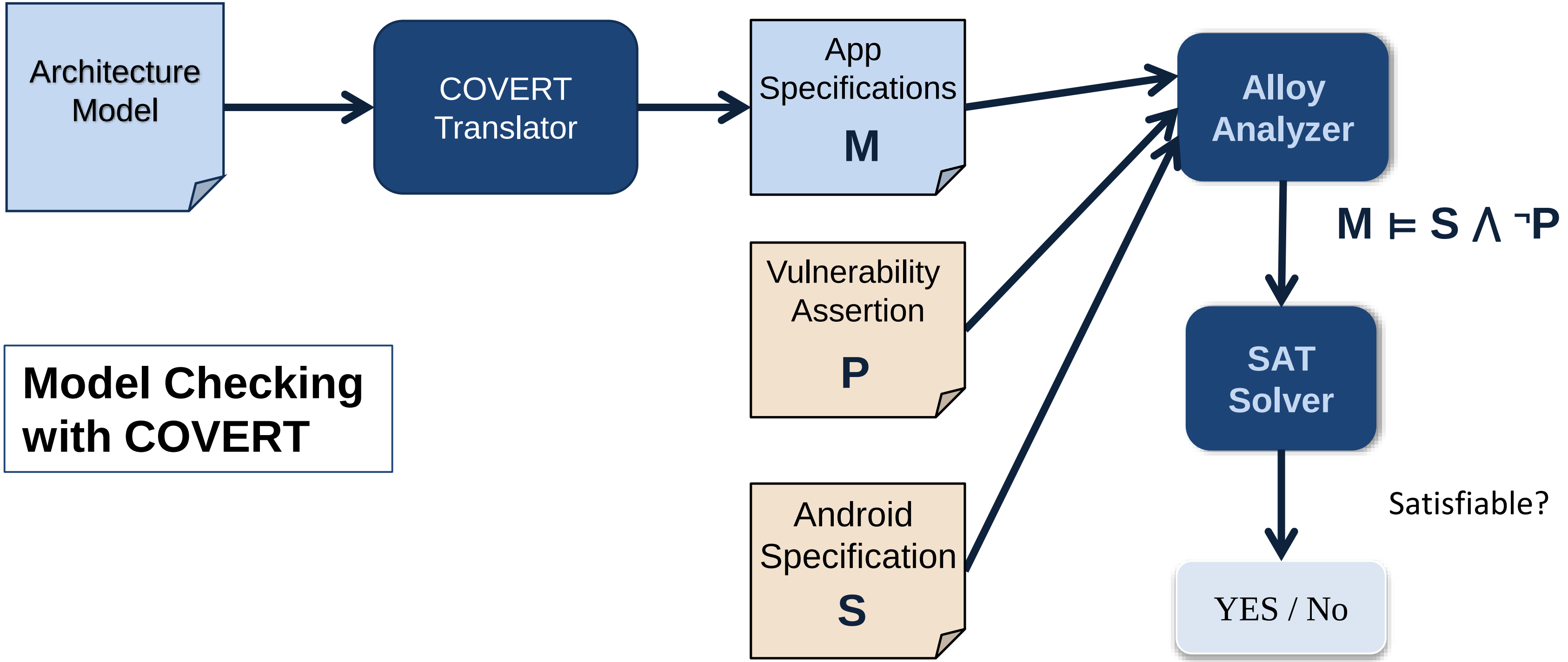
Many security violations come from data flows between components

Some of these can be checked via first order predicate logic

- E.g., Annotate applications with trust levels and check information disclosure
  - ```
    forall a1 :! AndroidApplicationGroupT in self.GROUPS |
        forall a2 :! AndroidApplicationGroupT in self.GROUPS |
          ((a1 != a2) ->
           (forall src in /a1/MEMBERS:!ApplicationElement/PORTS:! IntentBroadcastPortT |
              forall activity :! ActivityComponentT in a2.MEMBERS |
                forall tgt :! IntentReceivePortT in activity.PORTS |
                  ((connected(src, tgt) and
                    contains(src.action, activity.intentFilters))
                      -> a1.trustLevel <= a2.trustLevel)));
    ```

Apps cannot communicate implicit intents to apps that have a lower trust level
    ```

Thrust 1: Recovering Android App Interactions



Model Checking with COVERT

Given Android specification **S,** app specifications **M**, and vulnerability assertion **P**, assert whether **M** does not satisfy **P** under **S**

Thrust 2: Discovering Dependencies by Tracking Issues

Observation: Dynamic dependencies are often revealed in co-change relationships.

Operationalization:

- We can mine issue repositories and revision-control systems to discover these "hidden" relationships.
- We can leverage this information to find architectural flaws among related sets of files.
- We have created a new architectural view, called "Issue Space", a sequence of Snapshots (Si):

IssueSpace = <S1, S2, … Sn>

where n is the # of commits in the revision history to address the issue

Thrust 2: Discovering Dependencies by Tracking Issues

- Each snapshot is a 2-element tuple:

Sr = <G, t>

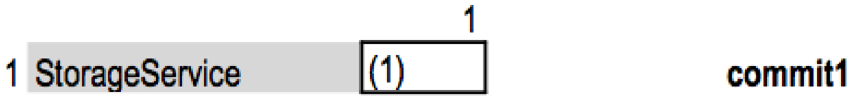
where t is the time-stamp when a commit is made to address the issue,

and G is a graph: <V,E>

where V is the set of files involved in the commit at time t.

Thrust 2: Discovering Dependencies by Tracking Issues

- Example: Apache Cassandra, Issue 436



Thrust 2: Discovering Dependencies by Tracking Issues

- Example: Apache Cassandra, Issue 436

		1	2	3	4	5	6	7	8
1	SuperColumn	(1)	Dp						
2	Column		(2)						
3	StorageService			(3)					Dp
4	SSTableScanner				(4)	Dp		Dp	
5	IteratingRow			Dp		(5)	Dp	Dp	
6	ColumnFamilySerializer						(6)	Dp	
7	SSTableReader	Dp	Dp	Dp	Dp			(7)	Dp
8	ColumnFamilyStore	Dp		Dp	Dp	Dp	Dp	Dp	(8)

commit2

Thrust 2: Discovering Dependencies by Tracking Issues

- Example: Apache Cassandra, Issue 436

		1	2	3	4	5	6	7	8	9	10
1	ReducingIterator	(1)									
2	SuperColumn		(2)	Dp	Dp						
3	Column			(3)	Dp						
4	StorageService				(4)						Dp
5	SSTableScanner					(5)		Dp	Dp		
6	ColumnFamilySerializer						(6)		Dp		
7	IteratingRow				Dp		Dp	(7)	Dp		
8	SSTableReader		Dp	Dp	Dp	Dp			(8)		Dp
9	QueryFilter	Dp,Impl	Dp						Dp	(9)	
10	ColumnFamilyStore	Impl	Dp		Dp	Dp	Dp	Dp	Dp	Dp	(10)

commit3

Thrust 2: Discovering Dependencies by Tracking Issues

- Example: Apache Cassandra, Issue 436

commit4

	1	2	3	4	5	6	7	8	9	10	11	12	13
1 ReducingIterator	(1)												
2 SSTableScanner		(2)						Dp	Dp				
3 ColumnFamilySerializer			(3)				Dp		Dp				
4 SuperColumn				(4)	Dp	Dp							
5 StorageService					(5)								Dp
6 Column					Dp	(6)							
7 ColumnFamily			Dp	Dp	Dp	Dp	(7)						
8 IteratingRow			Dp		Dp		Dp	(8)	Dp				
9 SSTableReader		Dp		Dp	Dp	Dp	Dp		(9)				Dp
10 Memtable			Dp	Dp	Dp	Dp	Dp		Dp	(10)			Dp
11 QueryFilter	Dp,Impl			Dp			Dp		Dp	Dp	(11)		
12 CompactionIterator	Dp,Ext	Dp	Dp				Dp	Dp	Dp			(12)	
13 ColumnFamilyStore	Impl	Dp		Dp	Dp		Dp	Dp	Dp	Dp	Dp	Dp	(13)

Ex:Extend; Impl:Implement; Dp:Depend

Thrust 2: Discovering Dependencies by Tracking Issues

Consequences:

- we have shown that when a system has files revised in many different issues—what we call a *hotspot*--these "shared" files are *connected*
- and that these hotspots almost always have design flaws leading to bugs, security flaws, and maintenance problems
- thus they should be analyzed and refactored

Thrust 2: Discovering Dependencies by Tracking Issues

- Example: Apache Cassandra

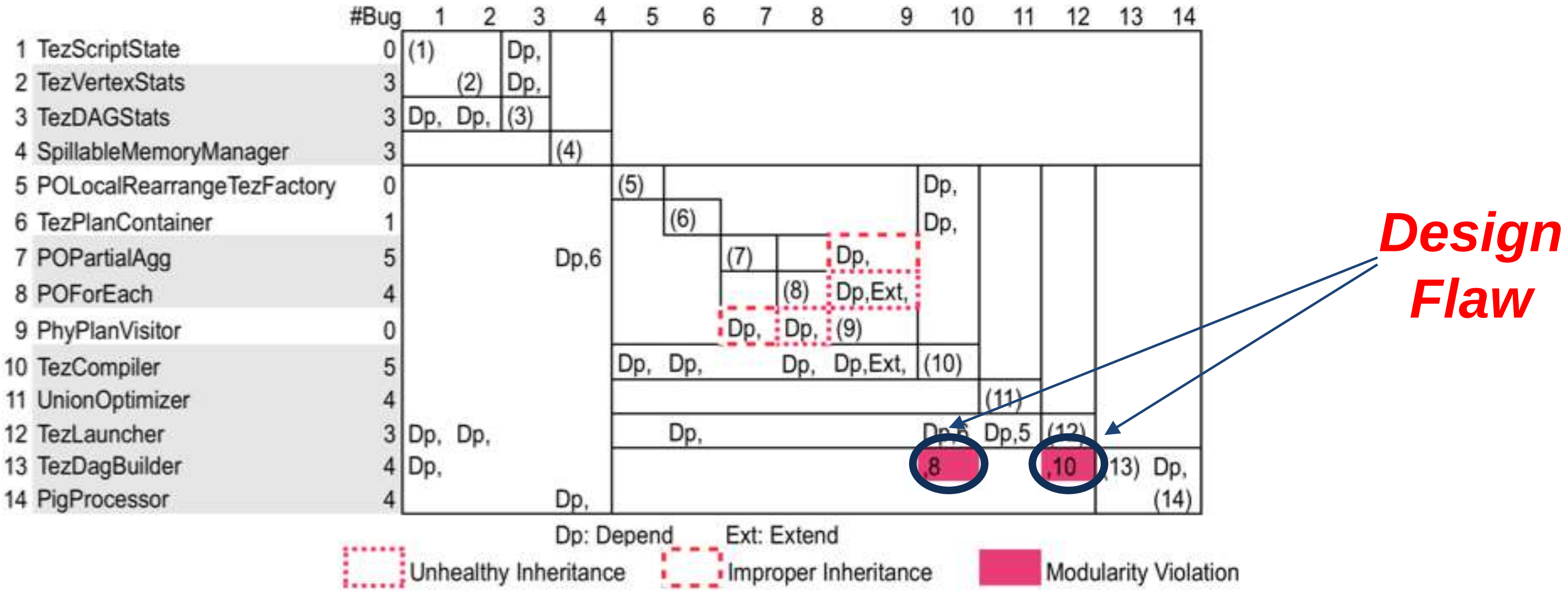
		1	2	3	4	5	6	7	8	9	10	11	12	13
1	ReducingIterator	(1)												
2	SSTableScanner		(2)						Dp	Dp		commit4		
3	ColumnFamilySerializer			(3)				Dp		Dp				
4	SuperColumn				(4)	Dp	Dp							
5	StorageService					(5)								Dp
6	Column					Dp	(6)							
7	ColumnFamily			Dp	Dp	Dp	Dp	(7)						
8	IteratingRow			Dp		Dp		Dp	(8)	Dp				
9	SSTableReader		Dp		Dp	Dp	Dp	Dp		(9)				Dp
10	Memtable			Dp	Dp	Dp	Dp	Dp		Dp	(10)			Dp
11	QueryFilter	Dp,Impl			Dp			Dp		Dp	Dp	(11)		
12	CompactionIterator	Dp,Ext	Dp	Dp				Dp	Dp	Dp			(12)	
13	ColumnFamilyStore	Impl	Dp		Dp	Dp		Dp	Dp	Dp	Dp	Dp	Dp	(13)

Ex:Extend; Impl:Implement; Dp:Depend

Design Flaw

Thrust 2: Discovering Dependencies by Tracking Issues

- Example: Apache Pig



Conclusions and Future Work

By considering dynamic information we can find design flaws, and hence locate the root causes of bugs more quickly.

This information is not available solely via static analysis; dynamic dependencies must be considered.

These flaws are the roots of technical debt.

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

Contact Information

Presenter / Point of Contact

Rick Kazman <kazman@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.

DM17-0780