**Week 5: Review and Analyze Conference Venue Review**

Nate Bachmeier

April 7th, 2019

TIM-8101: Principals of Computer Science

Northcentral University

# Describe the types of papers presented/research conducted at the venue

The International Symposium on Software Testing and Analysis focuses on software testing and best practices.

# Describe each track and types of papers that are submitted to each track

* Artifacts
  + A forum to discuss frameworks, libraries, and reusable data sets.
* Technical Papers
  + A forum for original and unpublished results
* Demonstrations
  + Discussions around tooling and concrete implementations
* Doctoral Symposium
  + A forum to discuss doctoral research with the community experts

# Paper 1: Analyzing the Analyzers: FlowDroid/IccTA, AmanDroid, and DroidSafe

## What problems did they solve?

Static taint analysis tools for Android applications are not compared against each other in a consistent manner (Qiu, Wang, & Rubin, 2018). A description of the configuration is rarely provided which further complicates reproducing the research results.

## What was their method?

A standardized benchmark and configuration environment were defined, and then tested against three popular open source products. This enabled an apples-to-apples comparison and empirical results.

## What are areas of future work/improvement?

Reflection and late binding, dynamic tracing. Inter-component communication

## What other works does this expand?

The researchers identified seven different comparison studies of these tools and found that the results could not be reproduced.

## Why is this important?

Future research is based on previous research, which must be accurate. If it turns out that apples to bananas comparison took place the research chain direction is flawed.

# Paper 2: Compiler Fuzzing through Deep Learning

## What problems did they solve?

Finding crashes in compilers is a complex and tedious task as they support vast combinations of input. They also tend to have very large code bases results in large sample sizes to gain enough code coverage. Fuzzing is challenging due to the highly structured nature of software code.

## What was their method?

They framed the question as a language modeling question, by sampling lots of open source code they can ask the computer to generate more code that looks like this. The generated files were then lint checked by LLVM and routed a subset sent to the OpenCL compiler collection.

## What are areas of future work/improvement?

Generation of highly structured random data can be useful in other automated testing scenarios. For example richer payloads can be generated to cover more code coverage.

## What other works does this expand?

CSmith is a state-of-the-art program generator for emitting random programs.

## Why is this important?

Previous efforts focused on hand optimizing the generators which required expertise and the end results were not portable. The deep learning approach only knows about the token sequences that are likely to appear next to each other. This makes it more adaptive to any programming language.

# Paper 3: PerfFuzz: Automatically Generating Pathological Inputs

## What problems did they solve?

Determining quality payloads for performance testing an application is complex and tedious. The tool uses mutation testing driven by coverage analysis to find the maximum stack traces possible.

## What was their method?

The system starts by performing a mutation on an existing user defined sample, and then measuring the code coverage involved in processing the input. The depth of the execution chain is treated as the reward function to the mutation engine. As it continues to seek out the maximum path in the application it will naturally find hotspots.

## What are areas of future work/improvement?

If the sample corpus is not sufficiently diverse then entire sections of the application can be missed. This results in the need for domain expertise to still exist in the performance test. Mechanisms need to exist to discover the internal data contracts and remove this complexity.

## What other works does this expand?

AFL is a code coverage driven fuzzer, which tried to hit all paths not focus on longest possible path. SlowFuzz is another tool that was compared but also faces the limitation of not being longest path centric, making it less likely to find the worst case input

## Why is this important?

Security based fuzz testing is more concerned with touching every single line at least once. However, performance testing is interested in the number of times they can hit a line within the same trace. This is because bottlenecks tend to occur in loops.