**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

## ISCE: Addresses the How

The International Conference on Software Engineering focuses on software engineering trends and explores the implications of those decisions. This is the premium conference where IEEE journal entries are discussed.

## MSR: Addresses the Which

The Mining Software Repositories focuses on analyzing and extracting metadata from software repositories. Many works center around GitHub and empirical studies of code hygiene.

## FSE: Addresses the What

The European Software Engineering Conference and Symposium on the Foundations of Software Engineering focuses on “recent innovations, trends, experiences and challenges in the field of software engineering (ESEC/FSE, 2018).” There are topic overlaps with ISCE and special attention to the application of software engineering.

## ASE: Addresses the Tedious

The Automated Software Engineering conference focuses on tooling and frameworks for simplifying engineering tasks. A large emphasis is placed on removing redundant operations.

## ISSTA: Addresses the Correctness

The International Symposium on Software Testing and Analysis focuses on software testing and best practices. The target audience is software development engineers in test (SDET) roles.

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

Every venue has a unique set of tracks that are available, but they tend to cluster into the same high-level themes.

## Technical, Research, and Journal-First Papers

The primary goal of this category is to present original and unpublished results that expand the knowledge of software engineering. Journal-First is a subset which provides a forum for recently published works that have not been widely demonstrated.

## Demonstrations, Showcases, Industry Use-cases and Artifacts

The primary goal of this category is to present concrete implementations of ideas that are being used in academia or industry experts. These tracks tend to focus on tooling, frameworks, and data sets built by the community.

## Doctoral Symposiums & New Ideas and Emerging Results (NIER)

The primary goal of this category is to share early investigation results as they pertain to ongoing research. Researchers can use the forum to have a dialog with the community experts and gain feedback on how to expand efforts.

## Challenges, Workshops and SE Education and Training (SEET)

The primary goal of this category is to provide hands on experience with a collection of tools and processes. Researchers will often perform studies using a specified data sets and their existing tools. This helps to have a consistent context and theme between the presentations.

# Describe Materials from ICSE

The selected materials were Magnus Frodigh’s Keynote, Code Review Comments Matter, and Are Code Examples Reliable.

## What problems do they solve?

The first contained a description of the vertical integrations that will be needed once 5G wireless becomes readily available. Frodigh describes Industrial IoT (IIoT) scenarios where a need for sensor networks will drive the need to innovate in data processing and storage technologies.

The other two focused of improving the quality of source code by addressing different aspects of the development process. As the efficiency of these steps are improved so is the productivity of the development team.

## What was their methodology?

The keynote approached the problem as an onion where one layer leads to the next challenge. For instance, he detailed the need for edge computing directly at the factory but also needing to blend into the cloud for cost efficiency. Then he continued to describe how these sensors will create tremendous volumes of data, which drives the need for improved AI algorithms.

The code review comments piece used natural language processing to determine the effectiveness of different language patterns. To determine the quality of online code examples, thousands of posts from Stack Overflow were parsed into an abstract syntax tree and their correctness measured against published best practices.

## What are future improvements?

The analysis of online code examples concluded that 31% of the Java samples had one or more issues. These issues were largely grouped into error handling, control flow, and incorrect API usage. I disagree with their findings, as the intent of online examples is to demonstrate invocation of a function. Adding error handling and conditional execution reduces the clarify of the example. It would however be interesting to see more investigation into the incorrect API usage scenario.

## What are related efforts?

The code review analysis was innovative as previous work has focused on sentiment analysis. While that is useful in many scenarios it does not work in code reviews as business policies explicitly request neutral emotional tone.

The quality of examples on Stack Overflow have been investigated in previous research. What made this approach novel was the parsing of the snippets into a custom syntax pattern. This allowed for many variations of the same code to clustered and compared as a single unit.

## Why are these works important?

Engineering time is expensive and needs to be efficiently used by providing the best feedback and the right examples the first time. This is difficult for the developer as they might lack the expertise, however algorithms can detect these issues and flag the issue upfront.

# Describe Materials from MSR

The selected materials were Public Git Archive, Anatomy of Functionality Deletion, and A Study on Inappropriately Partitioned Commits.

## What problems do they solve?

The first paper addressed the challenge of finding sufficiently large datasets for academia to perform Big Code analysis.

The second paper investigated reasons that mobile applications remove entire features. This is somewhat unexpected as it violates Lehman’s law of software evolution.

The third paper strategies to combine multiple related commits into a single logical developer task. This reduces the analysis time required to explore source code history.

## What was their methodology?

The Public Git Archive attempts to make a snapshot of the most popular repositories on GitHub. Each instance of the snapshot is approximately 3TB in size and contains both the code and commit metadata. This is accomplished by listening to public events from GitHub and then scripting the git tool chain.

The final paper determined the likelihood that two commits are related by first discovering all methods touched by the commit. Then using a graph representation of the source tree, they calculate a distance between the method sets. If the distance is within a threshold then the commits are said to be related.

## What are future improvements?

The researchers inspected commit messages to Android mobile manifest files across several hundred open source repositories. The manifest files were targeted as they declare all components exposed to the UI layer.

To understand why a feature was deleted the researchers focused on commit messages text. However, they also reported that nearly 60% of all commits did not contain a message. Instead of focusing on an individual file perhaps the research should focus on the source tree structure itself.

The primary reasons functionality was removed were (1) it didn’t work, (2) negative user feedback, and (3) not compatible with the supported devices. Using telemetry and crash reports could help to empirically drive decisions on when to pull the plug.

## What are related efforts?

Bundling a collection of software repositories into a single artifact is not a new idea and has been released in the past. Those artifacts have focused on niche areas such as only Java code or only mobile applications. The Public Git Archive is unique because it contains the most frequently book marked irrespective of repository contents. That results in a wider range of examples and can be leveraged across a more diverse set of objectives.

## Why are these works important?

When software fails the first question everyone asks is what changed? Having the ability to efficiently discover related commits and report them as a single unit reduces the complexity for source history analysis. Previous efforts relied on physically rejoining commits during the branch promotion process, which only works in a limited number of scenarios.

# Describe Materials from FSE

The selected materials were Darwinian Data Structure Selection, FraudDroid, and Microtask Programming.

## What problems do they solve?

Developers often compose software libraries with generic data structures in their default configuration. Tuning these parameters can significantly improve performance in some scenarios, yet it is rarely done as the effort is tedious.

Mobile applications expose advertisements as a mechanism to finance the development efforts. A subset of these applications uses fraudulent behaviors to trick the user into clicking on the ad or charging the advertiser extra.

Gaining participation on open source projects is difficult due to the learning curve required to make even simple changes. Instead a strategy is explored to boil software tasks down to 15-minute micro tasks which do not require broader context.

## What was their methodology?

The data structure selection was performed by first parsing the code into an abstract syntax tree. Then the generic definitions were detected and swapped out with various concrete implementations. The effectiveness of the transformation was measured with a user defined benchmark set. As improvements were discovered the system generates pull requests to update the originating code.

## What are future improvements?

Under the micro programming task model a software change is broken down into the smallest units possible. For instance, one microtask might be write the method signature, another to add the argument checks, and yet another to return a value.

This trades complexity between project management orchestration and development task implementation. Another challenge arises when the tasks need to be created in the first place. This requires a team of engineers with in depth knowledge of the application context. While it works in the contrived example more research on a real project is also needed.

## What are related efforts?

Previous ad fraud detection algorithms focused on either static analysis or network traffic monitoring. These approaches are insufficient as the development team is intentionally fraudulent. It is trivial to add obfuscation or encryption to the network traffic to cause false negative responses. Static analysis can also be fooled by deferred binding, such as Reflection. Instead the FraudDroid tool mitigates these scenarios by directly monitoring the UI view state tree at runtime.

## Why are these works important?

Darwinian Data Structures provides a powerful mechanism to enumerate through patterns and find efficient solutions. Without automated systems like this, teams are not going to invest their limited resources into tedious tuning.

These auto-tuning systems can be expanded to operate on additional patterns. Consider the scenario where a developer writes the software once and the compiler generates 10,000 different variants. Assuming each can be economically tested in parallel it provides a technique to build highly efficient code.

# Describe Materials from ASE

The selected materials were COBRA: An Interactive Static Code Analyzer, Mining Structures from Massive Text Data, and Software without Borders.

## What problems do they solve?

Static analysis across large code bases has both a large upfront setup cost and then runtime delay. COBRA addresses these latency issues by performing pattern searching across token streams.

The internet is full of unstructured data and software engineers need a collection of generic transforms to make the data semi-structured and consumable. The researchers provide a standardized pattern to add the structure and then extract insights.

Across a global infrastructure there are billions of Exception objects thrown. These need to be collected and triaged. Using automated processes, the industry expert describes how the entire investigative lifecycle is handled by the tooling.

## What was their methodology?

Adyen’s global web presence collects 1 billion Java Exception objects daily. A feature extraction process is then performed to cluster these exceptions into unique issues. For each issue stack traces are randomly sampled and orchestrated through a symbolic execution engine. The executor uses reinforcement learning to steer the symbolic executor into a reproduction of the failure. Finally, the system will open a defect for the development team with script that demonstrates the issue.

## What are future improvements?

COBRA parsing the source tree into a token stream, which is represented as a linked list structure. Some of the tokens represent the start of a scope block and contain a pointer to the close of the scope. This provides a mechanism to find token ranges in constant time.

The system was originally designed for C and later updated to support parsing Java and Ada source files. Since C programming does not have a native object support their tool was not designed to natively support inheritance search. Instead they must enumerate through all tokens in the source tree. There are likely other similar scenarios where an index of some means can further improve performance.

## What are related efforts?

Mining structure from massive text relies on the collaboration of many published research efforts. The author describes a framework to combine these disjoined efforts into a single workflow process.

The process begins with identifying words and then performing statistical operations to combine these into phases. Then topic mining is performed to find the categories of phases. Next entities are identified along with their type and co-type. Entities and topics are then attributed with metadata as richer context builds on richer context. The result of that effort is a multi-dimensional knowledge graph, which can be explored with standard graph traversal strategies.

## Why are these works important?

These works are important as they address common bottlenecks in the software development lifecycle. The less time developers need to spend poking around looking for defects the more time they can spend on new features.

# Describe Materials from ISSTA

The selected resources were Analyzing the Analyzers, Compiler Fuzzing, and PerfFuzz: Automatically Generating Pathological Inputs.

## What problems do they solve?

The first paper explores the performance characteristics of three Android static taint analysis programs. The researchers created a consistent configuration and benchmark to show that several previous result sets cannot be reproduced.

The second paper addresses the challenges of writing fuzz tests for a compiler. This is complex to generically solve as the generated code must adhere to numerous structural rules.

The third paper creates an algorithm for discovering the longest execution paths within an application. This enables it to discover the worse-case input scenarios and identify performance bottlenecks.

## What was their methodology?

Previous efforts attempted to fuzz test compilers have focused on building very sophisticated code generators. Instead the researchers decided to approach the problem as natural language scenario and use recursive neural networks (RNN) to learn the structure and emit tokens that were likely to be seen together.

The generated source files were first lint checked through LLVM before handed off to a collection of different compilers. These compilers were executed at different optimization levels and monitored for crashes or timeouts.

## What are future improvements?

The PerfFuzz tool generates random inputs for a program then measures the code coverage to determine the length of the execution path. The length is given to a reinforcement learning algorithm which searches for the maximum path. Frequently the maximum path will include a hotspot making this ideal for performance testing.

One of the challenges with this system is the need to provide initial seed input or the mutations can end up missing entire feature sets. Instead this approach should be combined with symbolic execution. That would allow the PerfFuzz to test various public method definitions without any domain context.

## What are related efforts?

In Analyzing the Analyzers, the authors noted that FlowDroid, AmanDroid, and DroidSafe are frequently compared in publish research. However, these publications do not specify the tool configuration used or consistently use the same benchmarks. This leads to apples-to-bananas comparisons which are not very useful.

The authors created a configuration that targeted the least common denominator across the three tools. Then they executed each test across the same benchmark set and manually verified the results. They found that the results varied wildly from previously published claims. Finally, they failed to explicitly reproduce the claims of six different papers.

## Why are these works important?

Finding software defects is a hard problem that is currently solved by throwing highly skilled humans at the problem. This does not scale to the needs of most organizations. Instead we need systems which can systematically find the issues for us. Having the ability to generate highly complex data structures, like source code, paired with maximum path solvers like PerfFuzz – really moves the needle and opens for those QA scenarios.