**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 paid 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 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The primary goal of this category is to present original and unpublished results that expand the knowledge of software engineering.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 by academia and industry experts. These tracks tend to focus on tooling, frameworks, and unique data sets.

## 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. Students and researchers can use this forum to have a dialog with the community experts and gain additional insights.

## 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. It is also common for a predefined data set to be provided and then researchers must gain new and innovative insights into it.

# 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?

Frodigh’s keynote described the vertical integrations that will be needed once 5G wireless becomes readily available. He centered the conversation around Industrial IoT (IIoT) scenarios where a need for sensor networks will drive the need to innovate in data processing, storage technologies, and improved AI algorithms.

The second resource explores the characteristics that are shared by highly rated code review comments. This could ultimately lead to best practices guidance for reviewers.

The third resource attempts to empirically discern the reliability of online code examples by comparing snippets against existing best practice documentation.

## What was their methodology?

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. Metadata was also recorded about the originating post such as the number of up votes it had received. This enabled correlations to be measured between accuracy of the post and the community scores.

## 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 don’t agree the findings are meaningful, as the intent of online examples are to demonstrate invocation of a function. Adding error handling and conditional execution reduces the clarify of the example. Ignoring these failure classes declines the error rate to around 12%

## What are related efforts?

To analyze the code review comments, natural language processing was used to study the language patterns. The code review analysis was innovative as previous work has focused on sentiment analysis (Efstathiou & Spinellis, 2018). That approach does not work on code reviews as many businesses require emotionally neutral tone. Instead the author focused on the use of suggestive words such as could, may, and should.

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

## Why are these works important?

Engineering time is expensive and needs to be used efficiently by providing the best feedback and the right examples the first time. Having algorithms that can detect these issues and flag the issue upfront improve the development experience and saves money.

# 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 why mobile applications remove entire features. This was unexpected as it violates Lehman’s law of software evolution, which states that software should only gain features overtime.

The third paper provides a generic algorithm to combine related commits into a single logical developer task.

## What was their methodology?

The Public Git Archive’s goal is to provide 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 to make a local copy.

The third paper determined the likelihood that two commits are related by first discovering all methods touched by a 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 inspected 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 evolution of the source tree structure itself can provide a more complete picture.

The primary reasons for removing functionalities were (1) it didn’t work, (2) negative user feedback, and (3) not compatible with the supported devices. Changing the deployment process can address issues (1) and (3) while (2) needs additional telemetry to drill into the details.

## 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 been scoped to niche areas such as only Java code or only mobile applications. Public Git Archive is unique in the way it contains the most frequently book-marked repositories irrespective of contents. This leads to a more diverse set of use cases.

## 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 of 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 reduce a developer task into a collection of 15-minute micro tasks. These micro tasks do not require broader context and are appropriately scoped for a new developer to immediately contribute.

## What was their methodology?

The data structure selection was performed by first parsing the code into an abstract syntax tree. Then the generic data structures are detected and replaced 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 fraudulent software can obfuscate or encrypt the network traffic to cause false negative responses. Static analysis can also be fooled by late binding technologies, 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, software teams are not going to invest their limited resources into tedious tuning. These auto-tuning systems can also be expanded to operate on additional patterns, perhaps driven by machine learning algorithms.

# 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 discovery of the worse-case input scenarios and similar 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 tool should be combined with symbolic execution. That would allow the PerfFuzz to start at arbitrary 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.

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.

## Why are these works important?

Finding software defects is a hard problem that is currently solved by tasking 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.

# 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 a large code base has both a large upfront setup cost and then runtime delay. COBRA addresses these latency issues by querying an internal representation and returning in seconds.

The internet is full of unstructured data and software engineers need a repeatable process to add structure, so their applications can consume it. The presenter details a standardized workflow accomplishing this task.

Across a global infrastructure there are billions of Exception objects thrown. These need to be collected and triaged. Using automated processes, an industry expert describes how the entire defect discovery and root cause analysis can be handled by automated 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. After the root cause analysis is detected a defect is opened and an example script attached.

## 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.

While the tool can scale to large code bases it might run into issues scanning something on the order of Debian’s package source repository. To support these massive code repositories there needs to be support for indexes into the linked list.

## What are related efforts?

Mining structure from massive text relies on the collaboration of many published research efforts. The presenter 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 words into phrases. Next topic mining was performed to cluster the phrases. Then entities are identified along with their type and co-type. Entities and topics are then attributed with metadata recursively as rich context allows the discovery of more context. The result of this process is a multi-dimensional knowledge graph. It can be explored with standard graph traversal strategies.

## Why are these works important?

COBRA and automated root cause analysis are important as they address common bottlenecks in the software development lifecycle. The less time developers need to spend hunting defects, the more time they can spend on new features.

# References

Arima, R., Higo, Y., & Kusumoto, S. (2018). *A Study on Inappropriately Partitioned Commits.* Retrieved from Osaka: https://sdl.ist.osaka-u.ac.jp/pman/pman3.cgi?DOWNLOAD=419

Basios, M., Li, L., Wu, F., Kanthan, L., & Barr, E. (2018). Darwinian Data Structure Selection. *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the FoundationsofSoftwareEngineering(ESEC/FSE’18),*.

Cummins, C., Petoumenos, P., Leather, H., & Murray, A. (2018). Compiler Fuzzing through Deep Learning. *ISSTA’18, July 16–21, 2018, Amsterdam, Netherlands*.

Dong, F., Wang, H., L, L., Liu, T., Xu, G., Guo, Y., & Klein, J. (2018). FraudDroid:AutomatedAdFraudDetectionforAndroidApps . *ESEC/FSE ’18, November 4–9*.

Efstathiou, V., & Spinellis, D. (2018, May 30). *Code Review Comments: Language Matters.* Retrieved from arXiv: https://arxiv.org/pdf/1803.02205.pdf

ESEC/FSE. (2018). *ESEC/FSE 2018.* Retrieved from Researchr: https://conf.researchr.org/home/fse-2018

Frodigh, M. (2018, May 30). *Live from ICSE: Conference Opening + Keynote from Nagnus Frodigh.* Retrieved from YouTube: https://www.youtube.com/watch?v=cpeMmMh7Syk

Han, J. (2017, November 1). *Mining Structures from Massive Text Data.* Retrieved from YouTube: https://www.youtube.com/watch?v=jy-Rg9r2lBQ

Holzmann, G. (2017, October 31). *COBRA - an Interactive Static Code Analyzer.* Retrieved from YouTube: https://www.youtube.com/watch?v=mVkz-ETpb0w

ICSE. (2019). *ICSE 2019 Paper Presentations.* Retrieved from ICSE Conferences: https://2019.icse-conferences.org/track/icse-2019-Paper-Presentations

LaToza, T., Lecce, A., Ricci, F., Towne, B., & Van der Hoek, A. (2018). Microtask Programming. *IEEE Transactions on Software Engineering* .

Lemieux, C., Sen, K., Padhye, R., & Song, D. (2018). PerfFuzz: Automatically Generating Pathological Inputs. *Proceedings of 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*.

Markovtsev, V., & Long, W. (2018). *Public Git Archive: a Big Code dataset for all.* Retrieved from arXiv: https://arxiv.org/pdf/1803.10144.pdf

Nayebi, M., Kuznetsov, K., & Chen, P. (2018). *Anatomy of Functionality Deletion.* Retrieved from UCAL Gary: https://www.ucalgary.ca/mnayebi/files/mnayebi/anatomy-of-functionality-deletions.pdf

Qiu, L., Wang, Y., & Rubin, J. (2018). Analyzing the Analyzers: FlowDroid/IccTA, AmanDroid, and. *ISSTA’18, July 16–21, 2018, Amsterdam, Netherlands*.

Van Deursen, A. (2017, November 2). *Software Engineering without Borders.* Retrieved from YouTube: https://www.youtube.com/watch?v=BxizjBmHXdA

Zhang, T., Upadhyaya, G., Reinhardt, A., Rajan, H., & Kim, M. (2018). Are Code Examples on an Online Q&A Forum Reliable? *Proceedings of ICSE ’18: 40th International Conference on Software*. Retrieved from UCLA.