**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 Mining Software Repositories focuses on analyzing and extracting metadata from software repositories.

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

There were three tracks for the 2018 MSR conference:

* Technical Papers
  + A forum for original or unpublished results that expand mining
* Mining Challenge
  + Researchers apply their tooling across a predefined data set and report their findings
* Data Showcase
  + A forum to discuss new and important data sets with the community

# Paper 1: Public Git Archive

## What problems did they solve?

One of the challenges to source code mining is knowing which repositories are available in the first place (Markovtsev & Long, 2018). The researchers identified the top 182,000 bookmarked repositories on GitHub.

## What was their method?

Public events are emitted as changes occur within a GitHub repository public. The researchers subscribed to these event streams and queued “git clone” operations across a farm of Docker containers. The cloned state and source code were then appended to a local copy.

## What are areas of future work/improvement?

There will likely be a large corpus of research which digs into the contents.

## What other works does this expand?

There are existing data sets which focus on niche scenarios, such as Java UI projects. The size of these lists is also relatively small which makes it difficult for

## Why is this important?

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 because it contains the most frequently book-marked repositories irrespective of contents. This leads to a more diverse set of use cases.

# Paper 2: Anatomy of Functionality Deletion - An Exploratory Study in Mobile Apps

## What problems did they solve?

Unlike classical desktop software, mobile software frequently removes entire features between releases. Empirical data into this phenomenon was not readily to explain the behavior (Nayebi, Kuznetsov, & Chen, 2018).

## What was their method?

Android mobile applications expose a manifest file which lists all components contained within an APK. The researchers examined changes to this file across hundreds of GitHub repositories over time. They identified the commit messages associated with code deletions and categorized them into reasons.

## What are areas of future work/improvement?

One of the challenges with this approach is that it could not determine a reason for nearly 60% of impacted change sets. An alternative approach is needed perhaps inspecting the source tree as it evolves over time.

## What other works does this expand?

The researchers suggest that most existing research has focused on finding examples of Lehman’s laws of software evolution. Their study is unique as it attempts to find the inverse.

## Why is this important?

The primary reasons for removing code are compatibility, negative user feedback, or flat out does not work. This suggests that as we transition to models of software as a service, other of laws of classical engineering may also stop being relevant.

# Paper 3: A Study on Inappropriately Partitioned Commits

## What problems did they solve?

Ideally there is only one commit per developer task, as this makes it clear why the change occurred. If a developer task is performed over multiple commits, then it is said to be partitioned. Partitioned commits are difficult to analyze as a clear mapping between the partitions does not always exist (Arima, Higo, & Kusumoto, 2018).

## What was their method?

The source code within the repository was first parsed into a graph structure. Next a mapping was created between each of the commits and the methods (nodes) that were impacted. Finally, the likelihood that two commits are related was calculated by distances between contained nodes.

## What are areas of future work/improvement?

The tests were performed across two open source repositories over a limited period. Using a larger sample size could identify scenarios with higher false positive rates. The authors did not provide any evidence all partitioned commits were detected.

## What other works does this expand?

Previous research attempted to find these situations and automatically untangle them into a new single commit.

## Why is this important?

Physically recombining partitioned commits is difficult especially in higher branches of code promotion. Having a mapping file between the commits removes that complexity and provides same behavior.

# 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

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