Analyze Conference Venue

Yiqiao Yin
W.Y.N. Associates, LLC

Abstract

This assignment investigates and reviews the top software engineering conferences. These conferences include ICSE, MSR, FSE, ASE, and ISSTA. For each of the listed conferences, the report reviews the main contributions and components of that conference.

1 ICSE

1.1 Brief

The International Conference on Software Engineering or ICSE is an annual international conference focusing on advanced research in software engineering. The first ever ICSE meeting was scheduled in Washington DC, USA and it was sponsored by the National Bureau of Standards and IEEE Computer Society. The ICSE also has an A rating from the Rankings of the Computing Research and Education Association of Australiasia.

1.2 Tracks

The ICSE has the following tracks: Technical Track, Software Engineering in Practice of EIP track, Joint Track on Software Engineering Education and Training or JSEET, software Engineering in Society or SEIS, New Ideas and emerging Results or NIER, Demonstrations or DEMO, ACM Student Research Competition or SRC, Artifact Evaluation or AE, Doctoral Symposium or DS, New Faculty Symposium or NFS. Each track has keynote speakers, posters or papers, presentations, and awards. This is very well funded and thoroughly organized on an international level. It would be a great honor if one gets their paper or work selected by ICSE.

1.3 Selected Work: Gopinath et al. (2014)

An important field in software engineering is the code testing. The work of Gopinath et al. (2014) discussed the existing problem of code test and search of the potential faults in a software program. In a perfect world, it is desirable to have the measure set up to detect the exact the faulty component in a code structure. The authors' work has two major motivations. The results needs to be applicable on large-scale set of real-world software programs and the programs in the database need to be diverse enough to be able to allow sufficient inference and generalization. The second motivation is to achieve statistically significant result. The proposed mutation testing delivers high performance. A long term goal that increase the desirability of mutation test is to develop alternative methods to detect software faulty components with the state-of-the-art accuracy. The

author's vision is quite forward. However, it is extremely challenging to keep this promise.

1.4 Selected Work: Inozemtseva and Holmes (2014)

The above research proposed mtuation test to serve as a test suite to detect faulty programs. In the same field, another team presents novel research of the relationship between test suite size, coverage and effective Inozemtseva and Holmes (2014). Their work seeks the answers to understand whether the test suite is correlated with a test case in the suite. In addition, their work also explores the ground of the effective of a test suite with regards of the statement coverage. In their work, the first contribution is a comprehensive survey of the investigation focusing on the correlation or association between coverage and effectiveness. According to their study, there is evidence show that there exists a low to moderate level of correlation between coverage and effectiveness assuming that the suite size is controlled. The value of their work is to initiate future research that can confirm the findings in real faulty programs that poses threat to validity.

1.5 Selected Work: denaro2015dynamic

Another interesting field is text generation. In this field, research in data flow testing has drawn a lot of attention in recent years. The classical data flow test which is static, however, fails to assist the software programmers to identify the relations, because the code flow is more dynamic than before. Hence, the work by Denaro et al. (2015) proposed a novel method called DynaFlow, which is new approach to generate test cases that takes the advantage of the useful data flow information where this was originally difficult to compute. To work with not just general programs but also interprocedural testing where some programs are object oriented programs, the design of DynaFlow is proposed to tackle dynamically allocated data flow.

2 MSR

2.1 Brief

The Mining Software Repositories or MSR holds its significant role in the field of data science, machine learning, and artificial intelligence in software engineering. The purpose and the motivation for MSR is to improve the development of software engineering practices using vast amount of data. These practices can involve source control systems, defect tracking systems, code review repositories, and so on. The conference encourages novel ideas in software development utilizing data science and machine learning, which is quite important at today's economy.

2.2 Tracks

MSR has many representative tracks. These tracks include technical work and research, mining challenges, and data showcase. In addition, hackathon is hosted on an annual basis to improve the development of the cybersecurity. The conference, just like ICSE, also offers keynote speakers as well as awards for each track session to encourage and incentivize scholars to make distinct contributions in the field.

2.3 Selected Work: Malhotra et al. (2021)

The work by Malhotra et al. (2021) investigate and compare the vast amount of feature reduction techniques in software change prediction. The field of software change prediction or SCP aims to explore the various structural metrics inside a software program and try to determine whether the future generation of the software is subject to changes and updates. The forecast or prediction conducted in the field largely depends on the correct specification of features which requires the researchers deep understanding and comparative analysis of different feature reduction or FR techniques to address the problems under high dimensionality, feature irrelevance, and feature redundancy problems.

2.4 Selected Work: Ali et al. (2020)

This paper is presented in the 17th MSR with a fancy title "Cheating Death" Ali et al. (2020). The authors developed a survival model to analyze the usage inactivity over a time span of 165 months on more than 3,000 python projects. It is the most interesting projects in this report and it is not surprising that the paper made it in the 17th MSR. The work has an extremely attractive motion. Computer scientists, data scientists, and computer engineers all rely on open source software or OSS which are public projects available for use. For machine learning and deep representation learning, it is common to use pre-trained models as a start. These projects tend to have longer survival time. However, for projects that are difficult to elaborate or to solve, it is challenging for developers to come back to the project and renovate it. For these projects, they are expected to have shorter survival time. The work done by this team shed lights to the effects of attributes of open source project and the overall length of activity level that an open source project could have.

2.5 Selected Work: Tufano et al. (2018)

Like the similarity score between two articles, the source code analysis provides similarity score between each component of code blocks an essential understanding of the software engineering tasks. This technique is important at clone detection, impact analysis, refactoring and so on. The similarity score or code similarity is a quantitative measure between two blocks of code from one or different software programs that consists of features automatically detected machines or hand-craft features by programmers. Recent development of deep learning or DL models can construct effect machines to replacement some of the cumbersome human labors. The work by the author Tufano et al. (2018) show state-of-the-art performance of a deep learning pipeline that detects and forecasts the similarity score.

3 FSE

3.1 Brief

Foundations of Software Engineering or FSE is a world renowned platform for researchers, scholars, practitioners, and educators to present and showcase their latest innovations and research practices. The conference consists of many different field and invite different scholars together every year.

3.2 Tracks

The conference starts with the plenary events and these events will have keynote speakers invited who are top of their fields. The conference, unlike the previously discussed ones, has a diversity and inclusion events to emphasize women in science and minority research groups which is quite important at today's world. The main events kick off with the main research track and industry track. The research track consists of work from academic or university labs while the industry track consists of work from labs in the industry. The conference also has a designated doctoral symposium for graduate students to showcase their ongoing research. The conference has a student research competition and workshop every year for junior level research.

3.3 Selected Work: Nguyen et al. (2014)

This work Nguyen et al. (2014) explores the dependence relations between application programming interfaces or APIs from online open source libraries. The central idea of the authors' work is to understand the preconditions of the APIs that would be used for ultra-large scale code structures with pre-defined built-in modules. To understand this task, the first step is to search for all client methods are used for invoking the APIs. Then it is essential to compute the potential conditions used in order to achieve the calls of certain libraries. Their work interpreted 120 million lines of code from the standard Java Development Kit or JDK library and delivered results with high recall and precision, 79% and 84% respectively. The paper sets important milestones of the idea of mining API references which is very important to today's coding standard considering that APIs are used everywhere in most of the components of today's world such as data science, machine learning, and deep learning. Based on the analysis, their work expects that the APIs to have preconditions that can make appearance in the large-scale corpus of open source libraries very frequently. Moreover, sometimes it is possible to predict the preconditions as well as the dependence relationship amongst different APIs before the library are called, which is very useful in software engineering.

3.4 Selected Work: Shi et al. (2014)

In large test suites, the regression testing is a fundamental tool that benefits most of the software develop environment. However, conventional regression testing suites share three difficult properties that pose challenges to move above and beyond. The first property is that the requirements of the test suites are governed by pre-defined coverage suites. The second property is that for scenarios where developers use reduced test suites the environment has to be tailored in order to satisfy all former requirements. The last property states that the quality of the software testing suites are described and written according to the reduced test environment. These properties posed great challenges to develop efficient software engineering environment for regression testing suites. The authors develop and explore the reduced test suites aiming to develop reduced test suites with properties such that they enable efficient development environment without having to be limited to the three properties.

3.5 Selected Work: Huo and Clause (2014)

This paper solves several issues in writing oracle machines. Oracle machines are another form of the Turing machine that is connected with an oracle. The oracle, in this case (not to be confused with the Oracle Script in Ancient China), is an entity of a black-box algorithm that is capable of solving certain problems. There are two major types of

problems: a decision problem and a function problem. A decision problem is the type of problems that is written as a set of natural numbers or a set of strings. Their solutions usually can be answered with "yes" or "no". A function problem is written by a function from a natural set of numbers. The solutions to function problems are usually value or some numerical output of a function of the set of numbers defined in the oracle. As a consequence, developers who are responsible for writing oracles or similar entities cannot always thoroughly check each step of the machine because the entity acts like a black-box in the pipeline. The resting test procedures are fragile and can be, sometimes, full of bugs. The debugging procedure is not that easy to handle either due to its dynamic structure or complicated developmental pipeline. The author proposed a technique that address these issues using two brittle inputs. The first type of brittle inputs are the type of assertions that has dependent relationship with uncontrolled inputs. The second type of the brittle inputs are the unused inputs which are the type of inputs that are not checked. These two types of inputs allow the users and the developers to automatically analyze test oracles.

4 ASE

4.1 Brief

The international conference Automated Software Engineering or ASE is a world-class research platform that is focusing on software engineering. The conference is held annually at a different location every year and it brings together the leaders and researchers of the world. The conference targets on topics such as analysis, design, implementation, testing, and software maintenance and so on.

4.2 Tracks

The conference has the following tracks. The main track, as a highlight, is the artifact evaluation, which focuses on evaluation of the most recent research in the selected topics such as software design, software implementation, and testing procedures and so on. Like the conferences above such as ICSE, MSR, or FSE), this conference also has a doctoral symposium to show case doctoral research and an industry showcase which provides opportunity for industry practitioners. Unlike the previous conferences, this conference has a specific gaming session called the ASE4Games.

4.3 Selected Work: Tantithamthavorn and Jiarpakdee (2021)

This paper points out the eXplainable Artificial Intelligence or XAI issues in software development. Specifically, the authors address the issues of injecting explainability in AI and ML models from the software engineering perspective. The purpose of deploying software engineering framework is to assist human decision making process. It is often times an ethical issue that needs to be solved instead of a technical issue. The success of AI-based assistant holds its crucial role at the center of the explainability power. The contemporary software development is a process of integrating, building, implementing many open source projects in an organic way. This process is often times checked by many software engineers while each carries a specific role and sets of knowledge to develop one block of code as a black-box without many questions. The authors showed that only 38% of the prediction studies came with global explanations and only 5% came with local ones. These results show lack of confidence that human users may have if are asked to deploy the models in production. To tackle this problem, the author used

LIME, which is a model-agnostic technique to explain the predictions of file-level defect prediction models. LIME is a great candidate because it uses linear boundary to establish local separation for explainability. The proposed approach is able to identify the tokens that is used to contribute to the overall software program. The explanation also allows programmers to debug more efficiently.

4.4 Selected Work: Liu et al. (2021)

The work by Liu et al. (2021) aims to explore and to predict tensorflow program bugs in real world developmental environment. The first glimpse the title show ambition and it is a big target to shoot for. Yet, it is quite necessary, because deep learning forms the fundamental of Artificial Intelligence or AI for most of today's technology. The most popular deep learning pipelines today are designed using TensorFlow from Google Brain or PyTorch from Facebook AI Research or FAIR. These two packages are well known and have very convenient APIs to design customized deep learning models. This paper investigates large-scale empirical data set of more than 12,000 failed TensorFlow jobs and examined the bugs reported in the TensorFlow programs. To tackle this problem, their work proposed a constraint-based approach to detect the shape-related bugs. The authors tested their proposed methodology on 60 industrial TensorFlow programs and showed that their work is efficient and effective. This branch of research is quite fruitful and can deliver promising results to the programs designed and developed in the industry. Because of wide variety of data sets that can appear in the industry, this branch of research is very attractive to the practice in the industry. However, the paper also indicated no false positives. This could pose as a threat to the robustness of the performance. The trade-off between false positives and false negatives is important to understand. In industrial pipeline, to ensure safety practice, it is much more risk averse to have a lot of positives cases identified, but some of them are false. This is not detrimental to the entire system, because the worst case scenario is the cost of going through the program and there is no harm to the entire system. The other case, false negatives, is much worse. If there are programs at fault and will harm the system and these are the programs undetected (shown as negatives), the reality will be detrimental and the result could harm the entire system.

4.5 Selected Work: Sartaj (2021)

The work by Sartaj (2021) has contributed a novel AI-based methodology to create test case scenarios for unmanned aerial systems or UAS. The motivation is straightforward for this branch of research, because the current UAS are evaluated on test case scenarios written by hand on a case-by-case basis. This way the pipeline is much more transparent, yet the procedure is cumbersome, inefficient, and inaccurate. The test settings are usually designed to check a handful of simulators at once. The author proposed a AI-based system to automate the UAS test case scenarios. The proposed technique is called AITester. It utilizes AI-based testing algorithms to create, generate, and evaluate test scenarios. The author shows preliminary results that the AITester is proficient at violating the expected behavior by autopilot.

Like all other AI-based programs, there is a trade-off between accuracy and transparency. This could lead to the next stage of research for AITester.

5 ISSTA

5.1 Brief

The International Symposium on Software Testing and Analysis or ISSTA is a famous leading research symposium focusing on software engineering, testing and analysis. Each year the ISSTA joins many different and diverse groups of people together from academia and industry to talk and exchange novel ideas. The conference aims to contribute to the community ground breaking research and new software systems that can be steering the direction of software production in the future.

5.2 Tracks

The tracks for ISSTA are simple and straightforward, which are much more easier to understand than the previous conferences. The technical papers hold the main spotlight of the conference each year. The conference also has workshops and artifact evaluations such as the other conferences. A unique track that other conferences do not have is the tool demonstrations. In addition, the conference also allows doctorate students to share their research as well.

5.3 Selected Work: Marinescu et al. (2014)

In practice, the software engineering production generates different level of source code during its evolution across different life time. The software is stored on a repository and the repository can be updated many times by a team or the project could involve more than one teams to use the same repository for development. In this environment, the version control and monitor serve a crucial role in this pipeline and currently the pipeline is exhaustive where different versions could take a lot of attention and cost to maintain. The author and his team proposed a new model called COVRIG. COVRIG is a flexible infrastructure designed to run each version in the system in parallel and create reports for each path that summarizes the performance metrics. It takes advantage of the empirical study conducted to investigate how programs evolve throughout the processes of code, tests and coverage.

5.4 Selected Work: Zhang et al. (2013)

Mutants in software engineering are syntactic transformations of the underlying program in the test environment. These mutants act as measure to allow the programmers to understand the given test at a certain measure, indicating whether there is a potential bug in the program. The concept "kill" refers to a situation where one program test certain results that come out to be completely different from the output in the original program. Denote the original program O and the mutant program for test to be M. It is said that the test T kills M if the test result using T on M is different from O. There is an underlying assumption in the argument stating that the program M is supposed to be the mutant of the original one, i.e. O. If this assumption is challenged, the argument would have to be reconstructed. This procedure of mutant test could be extremely costly if there is a large number of mutants. The author Zhang et al. (2013) proposed a family of methodologies that focus on reducing the test costs to ensure quicker and more efficient way of checking or potentially killing the unnecessary or incorrect mutants.

5.5 Selected Work: Loyola et al. (2014)

In the evaluation of a computer program, test data is created consisting of data inputs with features called oracles. The test data is then tested using a computer program or a programming system to make execution. The evaluation requires the test input data to be fed into the system to create test results that are same with test oracles. If they are different, then test oracles determine what is failed scenario in the final evaluation. The test oracles are originally pre-defined or created by hand based on agreed upon protocols amongst programmers. However, recent development has show novel direction that test oracles should be tailored to particular test inputs to ensure generalization across different programs and platforms. The authors seek novel direction to create automated test case and hence generate meaningful test scenarios for the oracle to create and to be used as the benchmark. It is believed that the testers, while using the proposed methodology, can improve the program with high performance and also return meaningful and interpretable results that can be customized to the real world situation. The proposed program is called DoDona which has a proposed algorithm to test input dataflow recordings. The authors show with empirical work of the performance of DoDona, which is able to improve the findings of up to 115% and reduce the search cost by 90%.

References

Ali, R. H., Parlett-Pelleriti, C., and Linstead, E. (2020). Cheating death: A statistical survival analysis of publicly available python projects. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 6–10.

Denaro, G., Margara, A., Pezze, M., and Vivanti, M. (2015). Dynamic data flow testing of object oriented systems. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 947–958. IEEE.

Gopinath, R., Jensen, C., and Groce, A. (2014). Code coverage for suite evaluation by developers. In *Proceedings of the 36th International Conference on Software Engineering*, pages 72–82.

Huo, C. and Clause, J. (2014). Improving oracle quality by detecting brittle assertions and unused inputs in tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 621–631.

Inozemtseva, L. and Holmes, R. (2014). Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th international conference on software engineering*, pages 435–445.

Liu, C., Lu, J., Li, G., Yuan, T., Li, L., Tan, F., Yang, J., You, L., and Xue, J. (2021). Detecting tensorflow program bugs in real-world industrial environment. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 55–66. IEEE.

Loyola, P., Staats, M., Ko, I.-Y., and Rothermel, G. (2014). Dodona: automated oracle data set selection. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 193–203.

Malhotra, R., Kapoor, R., Aggarwal, D., and Garg, P. (2021). Comparative study of feature reduction techniques in software change prediction. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 18–28. IEEE.

Marinescu, P., Hosek, P., and Cadar, C. (2014). Covrig: A framework for the analysis of code, test, and coverage evolution in real software. In *Proceedings of the 2014 international symposium on software testing and analysis*, pages 93–104.

Nguyen, H. A., Dyer, R., Nguyen, T. N., and Rajan, H. (2014). Mining preconditions of apis in large-scale code corpus. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 166–177.

Sartaj, H. (2021). Automated approach for system-level testing of unmanned aerial systems. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1069–1073. IEEE.

Shi, A., Gyori, A., Gligoric, M., Zaytsev, A., and Marinov, D. (2014). Balancing trade-offs in test-suite reduction. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pages 246–256.

Tantithamthavorn, C. K. and Jiarpakdee, J. (2021). Explainable ai for software engineering. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1–2. IEEE.

Tufano, M., Watson, C., Bavota, G., Di Penta, M., White, M., and Poshyvanyk, D. (2018). Deep learning similarities from different representations of source code. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 542–553. IEEE.

Zhang, L., Marinov, D., and Khurshid, S. (2013). Faster mutation testing inspired by test prioritization and reduction. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, pages 235–245.