**Research Statement**

Umar Farooq

Software has significantly contributed to humanity’s progress in almost every domain, especially after the wide adoption of programmable devices in the last few decades. From data centers and supercomputers to smartwatches and pacemakers, computing machinery and devices rely on software to drive their functionality. Given how software touches almost every aspect of our lives, software failures and inefficiencies cost billions of dollars annually. There has been lots of research on improving software reliability in traditional computing environments. However, the rapid evolution of smart devices such as smartphones has posed unprecedented challenges for software researchers and practitioners; the existing tools and literature have not caught up with the increasing ubiquity of mobile systems.

My primary research interests lie at the intersection of software engineering, programming languages, and mobile systems. I develop techniques that bring reliability and efficiencies to software in mobile systems and beyond. More specifically, my current research focuses on the following directions: automatically improving software reliability, leveraging machine learning for improving developers’ productivity, and improving software efficiency.

**Current Research**

**Improving software reliability** [MobiSys’18, OOSPLA’20]

Ubiquitous computing promises unprecedented connectivity and productivity. However, the ubiquity of computers brings new software runtime environments and introduces novel challenges; one such challenge is software reliability under these new environments. One of my research goals is to eliminate the reliability problems of mobile apps caused by the volatile runtime environment of mobile platforms (such as Android). Unlike traditional computing platforms (such as desktops and servers), mobile platforms provide a volatile running environment for apps where the state of an app may get destroyed and recreated frequently, either by runtime configuration changes (e.g., changing the phone orientation) or by limited resources available on the device (e.g., low memory). When the app state is not saved and restored appropriately, the app restarts may cause a wide range of runtime issues, ranging from loss of user progress and poor responsiveness to malfunctioned UI and app crashes. According to our recent study [MobiSys’18] on a large number of real-world mobile apps, a high percentage of them fail to handle restarts appropriately, posing a great challenge to mobile computing. To address the state issues caused by app restarts, we have explored two orthogonal directions: (i) a system-oriented solution and (ii) an application-oriented solution.

In the first direction, we aim to prevent apps from restarting by developing a restart-free runtime system, called RuntimeDroid, which completely avoids the needs of configuration-related app restarts. The purpose of restarts in the existing runtime system is to load resources for the new configuration (e.g., a layout for the portrait mode). RuntimeDroid achieves this by automatically identifying and “hot”-loading the resources while the app is running. The details of this work have been published in MobiSys’18 paper, which was well-received by the research community – it won the Best Paper Runner-up Award and was featured in the ACM SigMobile Research Highlights.

In the other direction, we approach the state issues with a more direct solution – first identify the necessary app state, then automatically preserve it at runtime. More specifically, we first designed a static analysis to systematically reason about the app source code and the associated resource files to find the critical program variables and properties of GUI elements that are necessary to be preserved. Then, we leveraged automatic code generation techniques to insert the state saving and restoring routines to the app code. We named this new solution as LiveDroid. According to our evaluation, LiveDroid can precisely identify and preserve the app state in real-world Android apps, which not only ensures the state correctness during app restarts but also makes the app restarts more responsive. The results of this work were recently published in OOPSLA’20, and the developed tools have been released on GitHub.

**Leveraging machine learning for improving developers’ productivity** [BigData’20]

Mobile app distribution platforms (such as Apple Store and Google Play) allow app users to post questions and reviews on their hosted apps and allow developers to make responses. Many studies have shown that promptly and satisfactorily responding to questions and reviews significantly improves the popularity of an app. However, a large number of questions and reviews make it virtually impossible for developers to respond manually. Although some efforts were made to build machine learning models that can generate responses to alleviate the burden of manually responding, existing models fail to generate app-specific responses, but rather generate generic responses learned from the limited pool of existing responses. Such responses seldom resolve user issues and thus are unsatisfactory.
Motivated by the above challenge, I initiated and led a collaboration with the data science group at the University of California, Riverside to build robust machine learning models for generating reliable app-specific responses. In this project, we built a generative machine learning model that is inspired by the work on machine reading comprehension (MRC) and that on natural language generation. Specifically, we fused MRC and sequence-to-sequence neural architectures in such a way that app-specific responses can be learned by merging information from app descriptions together with existing relevant responses. The experience of leading this collaboration has been immensely rewarding and productive. Our paper on this project was accepted for publication at the IEEE BigData’20.

Improving software efficiency [ASPLOS’19]

Finally, I was part of a team that developed a scalable and automatic compilation system that compiles JSON queries into parallel executables with bounded memory footprints, named JPSTREAM. The system uses a novel compilation technique that jointly compiles queries and JSON syntax into a single automaton. This compilation technique allows combining querying and parsing into one pass, which eliminates the need for constructing memory-heavy parse trees. Furthermore, JPSTREAM utilizes the “enumerability” property of automata to break dependencies and prune unnecessary query paths. Our evaluation of JPSTREAM shows that it consumes much less memory than existing parsing-based JSON tools. More importantly, it achieves near-linear speedup on multi-core processors, while the existing ones can only run in serial due to the inherent dependencies. This work was published in ASPLOS’19.

Future Research

Moving forward, I will capitalize on my background to explore emerging research topics in software reliability, and gradually steer my efforts towards exploring topics in software privacy and security, as well as leveraging machine learning techniques in software engineering. Some more specific directions are listed below.

Generalized application state preservation. I plan on making our work on app state detection more general by extending it to more programming environments and platforms. Interestingly, traditional managed programming languages share many properties and issues with mobile programming environments. Our findings and techniques in app state preservation provide foundations for improving app reliability under immature shutdowns. Specifically, I hypothesize that our app state detection techniques can be tuned towards widespread managed programming environments such as Java and JavaScript. Furthermore, I will explore the applicability of our techniques in emerging ubiquitous computing domains such as the Internet of Things.

Improving the analysis of apps written with multiple programming languages. Modern app development frameworks allow developers to employ several programming languages for building a single app. For example, in the Android development environment, developers can write parts of their apps in JavaScript, and other parts in Java. Unfortunately, existing analysis tools struggle to analyze the interactions among the components of hybrid code bases; existing tools typically focus on the constructs of one language at a time, leading to imprecise analysis. I plan to develop techniques for jointly analyzing hybrid code written with multiple programming languages.

Machine learning in software engineering. Recent developments in machine learning have made it possible for users to interact with computing systems in more convenient ways such as using voice commands. For a user to interact with an app using voice commands, the app must implement a natural language interface, which requires specialized skills and expertise at the developer side. Take Erica, the automatic assistant in Bank of America’s mobile app, for example. This assistant allows users to invoke certain features using natural language. Erica was built manually, and it is tuned specifically towards the Bank of America app. Such functionality is desirable in many other apps as well. I plan to build, with the help of my collaborators who are machine learning experts, a framework that automatically maps natural language constructs into invocations of app features.

Secure and privacy-aware runtime in IoT environments. In Internet-of-Things (IoT) applications, data transfer typically occurs across IoT devices, a mobile app, and the cloud, where multiple programming frameworks are involved. For example, a smart surveillance camera would send footage to the cloud through a paired smartphone. In such a scenario, the programming logic of transferring the footage requires to program both smart camera (e.g., SmartThings) and the mobile app (e.g., Android). Taint analysis techniques have been successful in reasoning about security- and privacy-related issues. However, the existing analyses only support a single programming framework; they cannot be directly applied to IoT scenarios where multiple programming frameworks are involved. I plan to build joint taint analysis techniques to unify IoT and mobile app program analysis by merging the control flow graphs of the involved programs based on their inter-device communication patterns.