**\chapter{Conclusions and Future Work}**

**\label{chap:conclusions}**

In this thesis, we attempted to address the problem of program understanding for JavaScript behavior within web application user interfaces. We argued that program understanding within the domain of front-end development was a challenge of mapping the page elements in the UI to the implementation in the source code. We proposed an approach to improve program understanding by modeling the mapping between the UI behavior in the browser view and the application logic in the code view. The key idea was UI behavior is dictated by mutations to the DOM elements on the page. Therefore, we needed to map the changes in the state on the page to the exact JavaScript statements that caused those changes.

We provided a way to visually inspect the DOM elements in browser view and link those elements back to the DOM mutators in the source code. We accomplished this by recording call-stack information about each DOM mutator during execution of behavior in the browser. The context information was captured in real-time and compiled into an execution history for the page. We provided insights into UI behaviors by modeling the DOM mutators as a control-flow graph. The DMG model provided semantic meaning about an arbitrary execution history by organizing DOM mutators based on the event handlers responsible for invoking them. We used JavaScript instrumentation to capture execution context data. That is, we injected our own analysis code into existing JavaScript source files as they were delivered to the browser.

We presented a programming tool called FireInsight to demonstrate our approach. To address interoperability we integrated FireInsight with the Mozilla Firefox browser as well as the Firebug web development tool. We evaluated our approach by applying our tool to a reference web application called JPS2.0. Using a series of detailed case-studies we showed how FireInsight could be used to improve developer understanding of the JavaScript behavior in the user interface of JPS2.0.

Thanks to the positive results from our evaluation of FireInsight, we feel there are a number of interesting directions to take our research. First, we need to investigate how to effectively detect the application-level event handlers when the source code integrates with JavaScript frameworks. As we saw for JPS2.0, FireInsight could not adequately detect application-level event handlers when Dojo source code was included in the JavaScript instrumentation. We bypassed the problem by explicitly ignoring Dojo source code during instrumentation. Today’s Ajax-driven web applications commonly use JavaScript frameworks or libraries to manage complexity. Therefore, it is crucial that FireInsight be able to handle analyzing code from JavaScript frameworks.

Second, we need to explore ways to make our JavaScript instrumentation procedure configurable, so that a developer can define what JavaScript statements will be captured by our analysis code. Currently we use a custom JavaScript parser that is implemented to look for very specific DOM mutators, such as calls to \verb!createElement()! or direct assignment statements. However, we do not account for all the possible methods for mutating DOM elements. Additional DOM API methods exist that can mutate page elements, such as \verb!appendChild()! and \verb!setAttribute()!. This should serve as motivation to evolve our system for JavaScript instrumentation so that FireInsight can be configured to analyze any arbitrary JavaScript statement.