

Research Statement for Mark S. Sherriff

My research interests lie in software engineering in general, but more specifically in development decision support, software reliability, and security. I am interested in analyzing and developing new methods and tools for guiding software development decisions, for estimating software reliability and security, and for finding better ways to incorporate these concepts into education.

Research Motivation

Software development managers balance lifecycle costs during development and maintenance with potential profits from sales. Releasing a software system early may gain an initial boost in profits by being first to market, but profit and customer confidence could easily disappear if numerous field failures are found due to a lack of poor quality. Conversely, delaying product release to remove minor faults may allow competitors to enter the market earlier and make these initial sales.

In industry, actual reliability of a software system also cannot be measured until it has been put into production and has been used extensively by the end user. Actual reliability information as found by the end user becomes available too late in the software lifecycle to affordably guide corrective actions to software quality. It is significantly more expensive to correct software defects once they have reached the end user compared with earlier in the development process. Because of this increasing cost of correcting defects and the need for software development decision support, software developers and managers can benefit greatly from information regarding any contributor to software reliability as early in the development process as possible.

My research addresses this need for earlier reliability information by investigating methodologies for detecting emerging areas of risk in a system by leveraging artifacts created during the development process and verification and validation techniques. By mining and analyzing data from source control system, test tracking tools, and defect management systems, we can provide guidance for defect removal, test case prioritization, and reliability estimation.

Analyzing Software Artifacts through Singular Value Decomposition to Guide Development Decisions

During development, programming teams will produce numerous different types of *software development artifacts*. A *software development artifact* is an intermediate or final product that is the result or by-product of software development. Some software development artifacts are created directly and intentionally by the development team, such as source code and design documents. However, other development artifacts are generated by the process itself, such as change records in a source control system, defect records, and test case logs.

Software artifacts created during development are often used for support purposes. For instance, change records might be referenced to track developer progress or to learn the current version of a file in the system. However, change records can also show how components interact with one another or how a system is evolving during development by examining what areas of the system change together and where these sets of changes are emerging. Testers could use information about change trends to direct their testing efforts whenever a new set of changes is created.

Using data mining techniques, software development artifacts can be used to identify *association clusters* in a software system that would not normally be apparent. An *association cluster* is a set of *code units* (e.g. files, packages, blocks of code, etc.) that exhibit a specific relationship with the other code units in the cluster through a particular development artifact. For example, association clusters created from file change records created from defect removal efforts will consist of files that, over a set of defects, tended to have to be modified together to repair a specific defect. Similarly, association clusters derived test case records would provide information about what areas of the code are being tested together. These association clusters could correspond to functional requirements, execution paths, or particular testing methodologies, thus exhibiting underlying aspects of the software system that might not readily be

apparent. Earlier research has shown that software development artifacts can be used in software change analysis to discover underlying structures within a code base.

The goal of this research is to build and investigate a methodology that uses software development artifacts to illuminate underlying relationships within a system that are not necessarily structurally dependent or could easily be detected through other analysis techniques to guide software development decisions. These underlying relationships appear as association clusters (i.e. bundles of files, directories, or blocks of code) that exhibit an affinity between themselves based upon the software development artifact that was used to generate the clusters. These association clusters in turn provide a type of “topographical overview” of the system that can be used to find other relationships between various areas of the project. For instance, areas of code that tend to change together should also be tested together and will also likely relate to specific requirements. If a set of field failures generate a specific set of association clusters that have no relation to any association clusters generated from testing data, then there is a possibility that there is a gap in test case coverage. However, these sets of changes might also include configuration files, help files, or other files that are not normally analyzed in an execution-based system analysis, such as a traditional impact analysis.

The methodology proposed in this research is called Software Development Artifact Analysis (SDAA). SDAA provides a framework for selecting and mining software development artifacts, generating association clusters, and then leveraging those clusters within the development process. Artifacts can be gathered from tools within the development process, such as change management systems, source control systems, and defect tracking systems. The data from these artifacts is compiled into a matrix that relates a particular code unit to other code units with regards to that artifact. A singular value decomposition (SVD) is performed on the matrix to generate the association clusters. The results of the SVD can then be utilized in various areas of development, such as impact analysis, test case prioritization, and program comprehension.

To examine the feasibility of SDAA, research was performed with an industrial project at IBM. Software development artifacts regarding changes initiated from defect removal efforts were gathered on three consecutive “fix pack” releases of an IBM product. We performed a short formative study on test case prioritization using the results of SDAA. Our results indicated that SDAA provided test case prioritization information that highlighted test cases that incorporated changed files and files that were candidates for subsequent changes.

Utilizing Verification and Validation Certificates to Estimate Software Defect Density

During software development, teams will use several different methods to make a system more reliable. However, the verification and validation (V&V) practices used to make a system reliable might not always be documented effectively, or this documentation may not be maintained properly. This lack of documentation can hinder other developers from knowing what V&V practices have been performed on a given section of code. If developers do not know where V&V has been used, extra time could be spent re-verifying an already thoroughly verified section of code, or worse, a section of code could go unverified. Further, this information could be used post hoc to see what V&V techniques were used on sections of code that have reported failures from customers. Using this failure information could help developers refine their V&V efforts for future projects.

A development team could benefit from a system that provided a means of V&V *evidence management*. In a software quality context, evidence management is a means of gathering the artifacts and other forms of evidence that a V&V technique was performed to improve V&V documentation efforts. This evidence can take the form of log files, written documentation, information in team management software, or anything else that records V&V effort. A software certificate management system (SCMS) can support this evidence management. A *software certificate management system* provides an interface and infrastructure to create, maintain, and analyze software certificates. A *certificate* is a record of a V&V practice employed by developers and can be used to support traceability between code and the evidence of the V&V technique used. *Our objective is to provide an automated method which allows developers to track and maintain a certificate-based persistent record of the V&V practices used during development and testing and to then leverage that V&V information to estimate defect density.* These V&V records could

also be used to improve the development process by monitoring V&V system coverage and providing a V&V reference for software maintenance and future projects. To accomplish this objective, we have developed **Defect Estimation with V&V Certificates on Programming (DevCOP)**. DevCOP is a SCMS which can be use for creating a persistent record of V&V practices as certificates. The DevCOP SCMS is implemented as a plug-in for the Eclipse integrated development environment (IDE).

Future Research Directions

My future research plans revolve around the ideas of development decision support, software reliability, and security and how they can be emphasized in both development and education. Finding effective ways to incorporate both of these concepts directly into the development process is a growing need as society becomes more and more dependent on software systems. Examples of poor reliability and security can be found in all facets of computing and now in everyday life.

SDAA Using Security-Based Development Artifacts – My dissertation work explores how SDAA can illuminate underlying structures in a system based upon the development artifacts it leverages. I will extend this work to examine artifacts generated specifically from security concerns and practices, such as checking for input validation. The association clusters from security-related artifacts could show potential attack paths through a system or sections of the system that are constantly under attack. These security records could be built into the system at compile time, providing users of this software (whether that is an end user or another system component) vital information at runtime about the various efforts put into this system to make it secure. This runtime security information could allow dynamic systems that require load balancing to place more sensitive tasks with the more verified system.

Developing Reliable Code – My current and past research deals with methods that can be used to help estimate reliability (through defect density) during the development process. However, I am also interested in ways to help developers design and code their systems to be reliable from their inception. Developers tend to circumvent good design and testing to meet a deadline placed on them, but often end up missing the deadline due to having to correct defects that they themselves introduced into the system. I plan to investigate what practices actually do contribute to the development of reliable code, and how one might be better than other. For example, it would be interesting to determine whether pair programming or a formal code inspection is better for building a reliable system. They are similar techniques in the regard that they both involve other programmers reading through another's code, but they are implemented in vastly different ways. With tool support, developers could easily record what sections of code were changed because of a given V&V technique. Over the development lifetime of a system, this information could provide insights as to what techniques work better in certain scenarios, and could assist developers by helping them target certain techniques for specific projects.

Educational Research in Reliability and Security – I feel strongly that Software Engineering is an important course for any CS student who is interested in pursuing a career in software development. To that end, and along with my interest in reliability and security, I plan to explore how reliability and security is currently taught to undergraduate students through their computer science education. Through this, I will analyze these teaching methods and find better ways to incorporate these important topics throughout the computer science curriculum. I will then empirically evaluate these new educational methods by looking at student comprehension and their ability to implement reliability and security concepts after the course has been completed. Our field is becoming relied upon by society more and more with each passing year. As with other forms of engineering, we need to ensure that our students are properly educated in how to make their products safe, secure, and reliable as we move forward in the twenty-first century.

In all of my research, my goal is to help practitioners, academics, and students alike realize the importance of software reliability and security and how it is affecting us all today. Software engineers need to understand that our field must be viewed to be as reliable as mechanical engineers or civil engineers because software is that prevalent in society. Through research and education, I know that I can contribute to these areas of our field.