

# Vilas Jagannath - Research Statement

I conduct research in the area of *software engineering*, focusing specifically on *testing* and *analysis*. The goal of my research is to develop techniques and tools to improve the efficacy of software testing. As software pervades our lives, the reliability of software assumes utmost importance. Testing is the most widely used methodology for improving software reliability and usually constitutes a substantial part of software development effort. My research deals with practical testing problems. My thesis research tackles challenges faced during regression testing of multithreaded code, focusing specifically on better expression of tests, and more efficient execution/exploration of tests. I have also conducted research on automated generation of complex test inputs, test repair, and testing/debugging of distributed systems. The research projects I have worked on have been published in top tier conferences (including ICSE, FSE, ISSTA, and ASE), invited for journal submissions (TOSEM and STVR), and received an ACM SIGSOFT Distinguished Paper Award (ICSE 2010). The projects have also resulted in the (open source) release of five frameworks/tools (<http://www.vilasjagannath.com/software>) that have been downloaded and utilized by both industry and academia. During the course of my research, I have collaborated successfully with faculty (from four universities), industry researchers, graduate students, and undergraduates.

In the future, I plan to continue to focus my research on software testing, debugging, and analysis. In the short term, I would like to continue my research on regression testing of multithreaded code. In the long term, I plan to tackle the testing challenges faced on mobile and cloud platforms, and develop better testing techniques/processes for security and privacy properties. The rest of this statement briefly summarizes my current and previous research followed by my plans for future research.

## Thesis Research

### Improved Multithreaded Unit Testing [3, 11]<sup>1</sup>

The advent of multicore processors has ushered in a new era of computing. Developers now need to write parallel code to fully utilize multicore processors. Shared-memory multithreaded programming is currently the dominant paradigm for writing such code. Unfortunately, as a result of non-determinism introduced by thread scheduling, multithreaded programs can display different behaviors for a fixed input. This leads to multithreaded programs often being afflicted by schedule-specific bugs like data races, atomicity violations, and deadlocks. The situation is worsened by the lack of a reliable, efficient, and intuitive method for writing and executing schedule-specific unit tests for multithreaded programs, which would allow developers to test for expected behavior under specific schedules and help prevent the occurrence of schedule-specific bugs. This is a grave concern since unit testing is an integral part of most testing processes and is relied upon to detect bugs early in the development process and prevent recurrence of previously fixed bugs.

---

<sup>1</sup>References listed in CV

```

1 @Test
2 @Schedule("[take]->add")
3 public void testBlockingTakeBeforeAdd() {
4     ArrayBlockingQueue<Integer> q;
5     q = new ArrayBlockingQueue<Integer>(1);
6     new Thread(
7         new Runnable() {
8             public void run() {
9                 @Event("add")
10                q.add(1);
11            }
12        }).start();
13     @Event("take")
14     Integer taken = q.take();
15     assertTrue(taken == 1 && q.isEmpty());
16     addThread.join();
17 }

```

Figure 1: Example IMUnit test

Currently prevalent approaches for writing schedule-specific unit tests involve the use of constructs like sleeps (real-time delays) or count down latches to implicitly specify and enforce intended schedules in tests. However, sleeps are an unreliable and inefficient mechanism for enforcing schedules and both sleeps and latches distract from the relevant logic of the test and do not provide good documentation of the intention of the test. To address these concerns, we developed IMUnit (pronounced “immunity”) which enables intuitive expression and reliable and efficient execution of multithreaded unit tests. Using IMUnit, developers can identify interesting points in the code as *events* and specify a *schedule* simply as a set of orderings over the events. During the execution of a test, IMUnit reliably and efficiently enforces the specified schedule, enabling testing for expected behavior under the specified schedule. Figure 1 shows an example IMUnit test which specifies a schedule where a blocking take operation in one thread is intended to occur before an add operation in another thread. We evaluated IMUnit by migrating over 200 existing sleep-based tests from various open-source projects to IMUnit tests. We found that IMUnit was expressive enough to specify the schedules that developers wanted to test, and that IMUnit tests were substantially faster (over 3x) than sleep-based tests. To help developers migrate their existing sleep-based tests to IMUnit tests, we developed automated migration support consisting of inference procedures with high precision and recall (over 75%). IMUnit has been released at <http://mir.cs.illinois.edu/imunit> and has already garnered considerable interest. It is being used by the Apache River project, and conversations are in progress with Guava (aka Google collections) and JSR-166 (`java.util.concurrent`) developers about using IMUnit.

## Change-Aware Exploration of Multithreaded Tests [4, 7]

Successful software evolves as developers add more features, respond to requirements changes, and fix faults. As software evolves, it needs to be (re)tested to ensure it has not regressed. This process, called *regression testing*, is used widely to improve the reliability of evolving software. However, as test suites grow over time, it becomes expensive to execute them against every new revision of the software. The problem is exacerbated when test suites contain multithreaded tests. These tests are generally long running since they need to be explored for many different *thread schedules* in order to search for bugs such as data races,

atomicity violations, and deadlocks. Note that even IMUnit tests, like the one shown in Figure 1, can be executed with multiple possible thread schedules, all of which satisfy the specified IMUnit schedule. While many techniques have been proposed for regression test prioritization, selection, and minimization for sequential tests, most exploration techniques for multithreaded tests do not consider regression testing (i.e., they only consider one version of the code).

We developed a novel technique, called Change-Aware Preemption Prioritization (CAPP), which uses information about the changes in software evolution to prioritize the exploration of certain schedules in a multithreaded regression test. We implemented CAPP in two frameworks for systematic exploration of multithreaded Java code: JPF, a stateful exploration framework developed by NASA, and ReEx, a stateless exploration framework developed by us. Using these implementations, we evaluated CAPP on the detection of 15 faults in multithreaded Java programs, including large open-source programs. The results show that CAPP can substantially (up to 5.3x) reduce the exploration required to detect multithreaded regression faults. We have released ReEx along with its CAPP implementation at <http://mir.cs.illinois.edu/reex>.

While CAPP deals with the regression testing scenario and performs *prioritization* of schedules, we have also developed another technique, called MuTMuT, which deals with mutation testing and performs *selection* of schedules. MuTMuT utilizes knowledge about the location of mutations within the exploration space to perform selection of schedules and substantially (up to 77%) improve upon basic mutation testing for multithreaded code. The MuTMuT paper was invited and accepted (with minor revisions) for publication in the Wiley STVR journal.

## Selected Other Research

### Generation of Complex Test Inputs [6, 9, 13, 14]

Generation of complex test inputs is one of the challenging tasks in testing. Manual generation of such inputs is tedious and error-prone. We have developed techniques to enable exhaustive (within bounds) and efficient automatic generation of interesting complex test inputs and demonstrated the efficacy of these techniques by applying them to find many bugs in complex software systems like refactoring engines and compilers. During my internship at Yahoo! Inc, I utilized some ideas from these techniques to generate interesting web server logs to test data pipelines that processed such logs to collect vital data. Our latest work in this area led to the development of UDITA, a language that allows users to combine the expressive strengths of both declarative and imperative test abstractions to create more expressive and composable test generation programs. This work was awarded an ACM SIGSOFT Distinguished Paper Award at ICSE 2010 and invited for submission to the ACM TOSEM journal. UDITA has been released at <http://mir.cs.illinois.edu/udita>.

## **Test Repair [5, 8, 15]**

Developers often change software in ways that cause tests to fail. When this occurs, developers must determine whether failures are caused by errors in the code under test or in the test code itself. In the latter case, developers must repair failing tests or remove them from the test suite. Repairing tests is time consuming but beneficial, since removing tests reduces a test suites ability to detect regressions. Fortunately, simple program transformations can repair many failing tests automatically. We developed ReAssert, a technique and tool that suggests repairs to failing tests code, which cause the tests to pass. Examples include replacing literal values in tests, changing assertion methods, or replacing one assertion with several. If the developer chooses to apply the repairs, ReAssert modifies the code automatically. Our experiments showed that ReAssert can repair many common test failures and that its suggested repairs correspond to developers expectations. ReAssert has been released at <http://mir.cs.illinois.edu/reassert>.

## **Testing/Debugging Distributed Systems [10, 12]**

In order to deal with data of immense scale, organizations are increasingly adopting distributed computing paradigms (e.g. cloud computing), and it is becoming more common for developers to write programs that execute on a large number of compute nodes. However, along with the greater computing power comes more daunting testing and debugging challenges. Since the computation is distributed over a large number of compute nodes, we need new testing techniques that are aware of the distributed computation scheme and new debugging techniques to help developers investigate failures. While interning at Microsoft Research, I helped develop a remote debugging and profiling technique for DryadLINQ applications that helps developers easily debug/profile the execution of their code on the distributed compute nodes. Prior to that, I also worked on mutation testing of Actor systems and helped define a set of mutation operators for such systems.

## **Future Work**

Testing is as old as programming itself, and testing will remain necessary and important as long as programming remains necessary. As I progress in my research career, I would like to continue to focus on software testing, debugging, and analysis with an emphasis on creating techniques and tools to help developers write and maintain tests better and improve the efficacy of testing. Within this focus, I envision extending my current/previous work in the multithreaded and distributed/cloud domains, and embarking on the new domain of security and privacy as described below.

## Regression Testing for Multithreaded Code

Despite great progress in research on testing multithreaded code, there is still no multithreaded testing tool that is widely adopted by programmers in their software development process. One key reason for this is the lack of integration with well-established regression testing processes. For example, when bugs are found in sequential code, developers usually write a unit test that fails in the presence of the bug and passes when the bug is fixed. This test helps developers reproduce and fix the bug. The test is also retained in the regression test suite to ensure future changes do not reintroduce the bug (i.e., do not regress). Unfortunately, such a process that allows developers to generate a regression test from bugs does not exist for multithreaded bugs. The reason for this is two fold. First, multithreaded bugs are usually schedule-specific and hence require schedule-specific tests to be reproduced. Second, once a multithreaded bug is fixed, the schedule that caused the bug may no longer be permitted, and thus may invalidate the test used to reproduce the bug. I would like to overcome these obstacles by first developing techniques to automatically generate XCTest tests that reliably reproduce multithreaded bugs and then developing a processes for retaining those XCTest tests as part of the regression test suite to prevent recurrence of the relevant bugs. Such research, focused on strengthening the testing processes used by developers, is vital for improving the reliability of multithreaded programs.

## Cloud and Mobile

As computing evolves, new platforms emerge with their unique advantages and challenges. Demands for increased mobility of both data and computation have resulted in the emergence of the cloud and mobile platforms which are now establishing their foothold in the computing landscape. These platforms entail not only new execution environments, but also new languages and frameworks. I would like to draw on my experience in the distributed computing domain and further research the unique testing challenges faced on these platforms and develop testing frameworks to help developers write effective tests for software that executes on these platforms.

## Security and Privacy

As software becomes ubiquitous, the implications of security and privacy flaws grow. While developer tests focus on functional correctness of software, non-functional requirements like security and privacy are usually tested separately by independent teams or during integration testing. However, in order to enhance awareness and detection of security and privacy flaws earlier in the development process, developers should be encouraged to write security and privacy tests along with functional tests. Towards this goal, I would like to develop frameworks that allow developers to specify and test for vital security and privacy properties effectively.