Documents on Research submitted by Abhik Roychoudhury

Email: abhik@comp.nus.edu.sg
http://www.comp.nus.edu.sg/~abhik

CONTENTS

Research Statement – articulated by Abhik Roychoudhury

Email: abhik@comp.nus.edu.sg
http://www.comp.nus.edu.sg/~abhik

I. Research Interests

Software testing and analysis,
Analysis and validation of real-time embedded software.

II. Brief exposition of what I research in

Software controlled devices and systems permeate our daily lives. The shift towards software controlled devices has taken place decisively for critical functionalities such as automotive and avionics control, as well as for common daily used ones such as television. In today's information technology age - software forms a critical asset of organizations, along with its more tangible assets like equipments, buildings and investments. More than the development of software - most of the effort in any large scale software project lies in the management and maintenance of it, as it ages.

In the past decade or more, my research has examined testing and analysis methods for software, with focus on real-time embedded software. In this context, I have examined and made contributions in three classes of methods – testing, debugging and analysis. In the following, I first provide a very brief exposition of software testing, debugging and analysis.

Testing: Software testing is an experimental method for checking software – checking it against designated inputs. The main issue is therefore one of choosing the appropriate inputs, so than even with incomplete experimentation of running the software with a limited set of inputs, one can obtain confidence in the overall software functionality. Several existing studies indicate that hundreds of billions of dollars are lost annually due to inadequate software testing infra-structure (a NIST study back in 2002 put the figure at 60 billion dollars only for USA).

Debugging: When software does not behave as intended – localizing the root cause of such unintended behavior manually is extremely tedious. The process of debugging refers to identifying the root-cause. It is commonly understood that the cost of maintaining a software system accounts for more than 90% of the project cost. Even though testing and debugging are widely adopted in industrial practice, many existing techniques are ad-hoc and this leads to a large number of post-deployment errors. The situation is exacerbated by the fact that software is increasingly used to control critical functionalities such as medical devices, robots, automotive and avionics.

Analysis: Finally, for critical software, it is crucial to develop a formal analysis of software properties to ensure predictable software behavior. For real-time embedded software, such an analysis could be used to predict non-functional properties such as execution time or energy consumption of the software. Since non-functional properties are dependent on the underlying computing platform, this also often involves an accurate modeling of the timing / energy effects of the underlying platform. Apart from giving guarantees about software behavior, such an analysis can also be exploited to generate representative test-cases, a thread of work pursued in my own research.

III. Research accomplishments – Software testing and analysis

In the following, I highlight some of the major research accomplishments in software testing and analysis. All of the works are post-PhD, and accomplished during my stay at NUS (2001 onwards) --- works done during my PhD study are not mentioned at all. For any line of work, the overall perspective and the visible impact are explicitly discussed. In particular, I have highlighted the major lines of work which were accomplished post-tenure (2007 onwards). Some of the notable pre-tenure works are mentioned briefly. For each pre-tenure work mentioned, its relationship and influence on any post-tenure work is also explicitly discussed.

A. Symbolic techniques for regression debugging and testing (post-tenure, 2008 – now)

In the past five years, I have worked on several novel approaches for testing and debugging, specifically with continuous software evolution in mind. Software engineering and formal analysis is a well-developed area in computing, and several testing, analysis and verification methods have been proposed in the past for enhancing software reliability. However, these works typically treat a program as a stand-alone artifact whose properties are analyzed. In reality, programs are continuously changing, with new features being added – as a result of which previously working features may no longer function properly. In software engineering, such a scenario is commonly called a *regression*. Finding the root-cause of such regressions (or debugging) is a major head-ache for software developers in *any* large development project!

The inherent difficulty in debugging lies in the lack of capture of intended software behavior. If the intended behavior is available in the form of a formal specification, one can locate the deviation of the actual observed behavior from the intended behavior, and thereby locate the root-cause. However, in real-life, programmers are reluctant to put in the extra effort for writing formal specifications. Since programming hardly occurs from scratch, we propose to use the previous working version of the program, as an informal specification of its intended behavior. In [FSE09], we use *symbolic execution* of past and current program versions to localize the root cause of software regressions. Symbolic execution corresponds to executing a program with symbolic or un-instantiated inputs, as opposed to concrete input values. This produces logical formulae which serve as a semantic footprint of the program execution. In [FSE10], we show how such logical formulae obtained from two implementations of the same specification can be systematically compared to find the root-cause of errors in either implementation. The applicability of such methods are very wide – apart from comparing program versions, they can be used to compare two implementations of the same protocol, such as two web-servers one of which is buggy. Alternatively, one can compare an embedded software against its non-embedded reference implementation, where the embedded software might have been optimized against resource usage. Indeed, we have employed our methods to debug the widely used embedded Linux Busybox against GNU Core-utils, or for debugging optimized web-servers against Apache. These show the scalability of our proposed debugging methods [FSE09,FSE10], and also their wide applicability.

In [FSE11, ICSE13-PRV], we have combined symbolic execution and program dependency analysis for summarizing behavior of program versions. We observe that along any program path, the output can be described as a symbolic expression in terms of the inputs. Our work automatically obtains a partitioning of program paths, such all paths in the same partition have the same symbolic expression for output. We show that such a partitioning can be effectively computed via on-the-fly calculation of dynamic program dependencies and slices. The main technical strength in the result lies in showing the completeness of our approach – where we show that all symbolic expressions for the output will be found by our method. Furthermore, for each such symbolic output expression, our method also reports the set of inputs which

will compute this output expression, with the set of inputs being given as a logical formula. As a result, we can generate test-cases --- for each symbolic output expression, we compute one test input which produces the symbolic output expression. Such a test input generation method is also found to be far more scalable than path coverage based testing methods, since a large number (or even unboundedly many) paths may produce the same symbolic output. We also show the use of our path partitions for software regression debugging in a scalable fashion.

Hand-in-hand with software debugging, we have started exploring the theme of automated program repair. Similar to debugging, automated repair stands to benefit from a crisp description of intended program behavior. In our work [ICSE13-SEMFIX], the intended behavior is extracted via an augmented symbolic execution of failed test cases. The extracted intended behavior then guides the fix generation which proceeds via constraint solving based program synthesis. Previous works on automated repair mostly proceeded by heuristic search methods such as genetic programming. Ours is the first work which repairs programs via semantic analysis. As a result, the repairs produced by our method are guaranteed to be correct, that is, they are guaranteed to fix the given program.

Apart from debugging and repair via symbolic execution, we have studied test generation methods to explicitly stress the effect of program changes in a systematic fashion. In [ASE10], we have developed a symbolic execution based method, which can construct an input to reach a program change, and then propagate its effect to the program output. In [FSE13], we have studied test generation to expose complex semantic interactions among program changes. Such change-interaction errors are found to be prevalent among well-tested and deployed software projects such as GNU Coreutils. Our work provides a basis for detecting subtle change interactions, and then generating test inputs which witness subtle change interactions.

Perspective: Symbolic execution serves dual usages in our proposed methods. First of all, symbolic execution alleviates the need for many concrete executions – thereby guiding search over a huge search space such as a space of paths. This usage is evident in our works on symbolic execution based regression testing – where symbolic execution guides the search over program paths, or sequences of program changes. Secondly, we use symbolic execution in another novel fashion – we exploit it to extract intended program behavior for certain test cases. This usage of symbolic execution is evidenced in our works on regression debugging and repair.

External Collaborations: The work [FSE09] was jointly with Microsoft Research, and the work [ICSE13-SEMFIX] was jointly with IBM Research.

Impact: The paper “DARWIN: an approach for debugging evolving programs” [FSE09] received the ACM SIGSOFT Distinguished paper Award, and was also show-cased at the Microsoft Professional Developer’s Conference (PDC) in 2009. This work has ~40 citations as of August 2013 (data from Google Scholar). The DARWIN work is now credited for being the first work in “formula based debugging” where logical formula are used describe program executions, and these logical formulae are analyzed for debugging. A talk by Prof. Alessandro Orso (Georgia Tech) which shows this pioneering role of the DARWIN work can be seen from

http://www.slideshare.net/alexorso/20130204dagstuhl-alex-orso (see slides 50-54)

Subsequently researchers in various places such as IBM, NYU, Berkeley, Max Planck and other places have started investigating in this area, as shown by the subsequent papers in the area. I have been invited to invitation-only events to share my perspectives on the use of symbolic techniques for software debugging. These include the Dagstuhl Seminar on Fault Prediction, Localization and Repair – Germany

(February 2013) and PAS 2012 International Seminar on Program Verification, Automated Debugging and Symbolic Computation - Beijing, China, organized by Chinese Academy of Sciences (October 2012). In February 2012, I co-organized a workshop on Future of Debugging to enable to greater academia-industry interaction in this area.

Furthermore, as a result of my expositions on this topic at many international venues and universities, I was recently appointed as *ACM Distinguished Speaker*. The ACM Distinguished Speaker program features "renowned thought leaders in academia, industry and government, speaking about the most important topics in computing and IT world." I am now undertaking lecture series in various universities and companies in my role as an ACM Distinguished Speaker. More details about the program is available from http://dsp.acm.org/

B. Dynamic dependency analysis and slicing (pre-tenure, 2003 – 07)

Current software debugging tools require active participation from the programmer to find the cause(s) of an observable error. This makes software debugging tedious and extremely time-consuming. It is well-known that developers routinely validate their programs by testing. However, if the output of a test case is unexpected, there does not exist suitable tools to automatically analyze the failed program execution for possible error causes. In this work [ICSE04, TOPLAS08], we developed JSlice, the first dynamic slicing tool for a full-scale real-life programming language (JSlice works over Java programs). Given an observable error found by program testing, it can perform automated program dependence analysis to highlight the probable error causes.

In a later embodiment of the work [ISSTA07], we also study the issue of bug report comprehension -- where we let the programmer guide the bug report construction. In other words, computation and comprehension of the bug report proceeds hand-in hand. This an important practical issue, since the comprehension of the bug report is often neglected by researchers as a less interesting post-mortem activity, but it is a huge issue with software developers.

Perspective: Apart from showing the scalability of dynamic slicing, and its applicability to a widely used programming language, this work also makes an important conceptual contribution. It shows that the analysis and dependency extraction needed for dynamic slicing can be achieved in the "compression domain", without decompressing the trace representation. Indeed, this is a key factor contributing to the scalability of our dynamic slicing method.

Impact: The JSlice tool reported in [ICSE04,TOPLAS08] has been used in over 300 organizations in 30 countries for teaching, research and development. The work [TOPLAS08] is a full version in a journal of the preliminary conference paper [ICSE04]. These works have 60+28 = 88 citations in total, as of August 2013 (data from Google scholar).

Influence on post-tenure works: My past work on dynamic slicing gave me a solid grounding on program dependency analysis. I have later developed novel combinations of dependency analysis and symbolic execution which has led to powerful methods for test generation, regression debugging or program summarization – as evidenced in the works [FSE10, FSE11, FSE13, ICSE13-PRV]. I believe such a combination of dependency analysis and symbolic execution methods allowed us to develop powerful testing and debugging techniques – which would not have been possible by using only symbolic execution or only dependency analysis.

C. Inter-object / inter-class behavior in software systems (post-tenure, 2007-12)

Communication between objects or classes in a software system can be captured via modeling notations such as UML Sequence Diagrams or their extensions. We have devised several modeling notations for capturing such inter-object communication either via a mix of intra-object and inter-object style specifications [TOSEM09], or via purely inter-object style specifications [FSE07, TOSEM12]. Furthermore, these notations have explored how similarly behaving objects can be symbolically represented, and how such symbolic specifications can be exploited for efficient model-based test generation. In more recent works [ICSE11, ICSE12], we have studied the dynamic inference of such inter-object specifications from software execution traces.

Perspective: Dynamic inference of models from execution traces is helpful for program understanding and is often useful in an industrial context – where a novice developer is told to take over the software written by other developers (who might have left the organization). Previous works had largely focused on inferring state-based models, or intra-object models. Our recent works [ICSE11, ICSE12] show that inferred inter-object style specifications are more concise and hence can aid program understanding.

Industrial Impact: The research in this direction obtained an IBM Faculty Award, which recognized the importance in industry of inferring this style of specifications / models for program understanding and maintenance. This also led to joint collaboration with IBM on suitable system modeling styles.

IV. Research accomplishments – Real-time and embedded software

In the following, I highlight some of the major research accomplishments in analysis and validation of real-time embedded software. All of the works are post-PhD, and accomplished during my stay at NUS (2001 onwards). Wherever applicable, I have also highlighted the works which were accomplished post-tenure (2007 onwards). Pre-tenure works are briefly mentioned.

A. Software timing analysis for modern embedded platforms (2001 onwards)

Timing is critical to the functioning of many computing systems. Indeed, real-time embedded systems are ubiquitous and control the functionality of many devices used in our everyday lives. Even safety critical application domains, such as modern day cars, are largely controlled by software. Many of such software is time-critical, that is, the execution time of the software is critical to its correct functioning. One primary example along such lines is the software controlling the brakes of a modern car. In this line of work, our aim is to develop methods which can estimate the Worst-case Execution Time (WCET) estimate of a given software – an upper bound of the execution time it can take for any possible input. Such an estimate should be safe (it should not underestimate the possible execution times) as well as tight (it should not greatly overestimate the actual execution times). WCET estimation methods have been studied for the past two decades and various methods / tools have been developed. The challenges we tackle are one of scalability – analyzing large-scale embedded software running on new generation environments/ platforms.

Pre-tenure work Our main effort in this direction is embodied by Chronos [SCP], a worst-case execution time analysis tool for C binaries. Chronos is an execution time analysis tool which can statically analyze C programs to predict safe and tight time bounds. The tool employs many technical novelties including the modeling of timing effects of many novel micro-architectural features [RTSS04, RTS05, RTS06, RTSS09-ucache] in the underlying processor platform. These include modeling the timing effects of processor

pipelines [RTSS04,RTS06], branch prediction [RTS06], and multi-level processor caches containing instruction and data [RTSS09-ucache]. We have also studied the timing effects of multiple tasks executing on the same processor core due to hardware resource sharing, such as cache. Such analysis computes the Cache related preemption delay [CODES03].

Post-tenure work: Most recently, in 2008-2013, we have studied the timing prediction of software running on multi-core platforms. Resource sharing in multi-cores, such as shared cache or shared bus, create timing interferences not present in single core platforms. Our work [RTSS09-sharedcache, RTSJ-to-appear, RTAS12] develops a combined shared cache and bus analysis. We employ path sensitive verification techniques are used in a controlled fashion to obtain accurate timing estimates, in presence of shared cache and shared bus. In parallel to studying analysis methods for bounding execution time of concurrent software on multi-core platforms – we have also studied the system-centric approach, where compiler controlled memories such as on-chip scratchpad memories are used to ensure predictable execution times for concurrent embedded software. In a scratchpad memory, memory blocks are pre-allocated – thus any given memory block is known to be either on-chip or off-chip. This is different from caches, where the same memory block may be hit at certain time, and miss at certain other times in program execution. We have developed worst-case execution time analysis guided compiler optimizations, such as on-chip scratchpad memory allocation for concurrent embedded software [TOPLAS10, LCTES11].

Apart from building a timing analysis infra-structure for software running on multi-cores, we have made other important advances to make timing analysis and prediction more practical and usable. Most of the works on worst-case execution time analysis consider a single monolithic execution of an application. However, in modern embedded platforms (including smart-phones) applications run on a supervisor software such as an operating system. Thus, the timing effects of system calls and interrupts need to be taken into account. In [RTSS11-OS], we have developed an analysis of system calls/interrupts in a widely deployed operating system, while in [RTSS13-OS] we have studied integrated timing analysis of multiple applications on a real-life operating system and processor hardware.

We have investigated various mechanisms to combining software timing analysis with other forms of validation such as – model checking [RTSS11-cache] or symbolic execution based test generation [RTSS13-cache]. These form important conceptual contributions in developing a general framework for analysis of quantitative properties of software such as timing or energy consumption. The work of [RTSS11-cache] improves the accuracy of abstract interpretation based cache timing analysis via repeated runs of model checking, while not exceeding a given time budget. Our most recent work of [RTSS13-cache] is in the novel domain of systematic performance testing – where abstract interpretation based cache timing analysis is used to guide symbolic execution based test generation. The test suite generated is guaranteed to cover all possible cache thrashing scenarios (for a given cache configuration) in a program.

We have investigated timing analysis of code generated from industrially used high level modeling languages such as Simulink / Stateflow, as well as for code generated from synchronous languages like Esterel. Unfortunately, when models are compiled into programming languages, the timing abstractions assumed at the model level may not be preserved at the code level. Previous research had proposed the use of specially designed processors to ensure that the timing abstractions at the model level (such as the perfect synchrony hypothesis) are satisfied at the implementation level. Our work [RTS12] shows that software timing analysis can be used on the generated code to guarantee the perfect synchrony hypothesis, thereby removing the need to construct special purpose processors. The generated code can thus be executed on off-the-shelf processors.

In terms of applications of timing analysis, we studied applications of our timing analysis infra-structure for integrated analysis of application and operating systems code [RTSS13-OS] to real-life robot controllers which periodically performs several tasks like balancing, navigation and responding to remote control. We also conducted an A*STAR funded thematic research project (2007 – 10) on embedded and hybrid systems for Body Area Networks (BANs). The central problem here is that certain elderly people may need to be monitored at home using sensors, since keeping them permanently in a hospital may lead to prohibitive costs. Thus, the health statistics of such elderly persons may be monitored via tiny sensors pasted on their body, as well a gateway device strapped to their waist which reports any alarms to the care-giver. We have used our timing analysis infra-structure to bound the time between monitoring of a patient's health and reporting to the care-giver [CODES08]. The analysis also determines the order in which the sensors on the person's body should be polled – so that the care-giver receives any alarm notifications in a timely fashion.

Impact and Collaborations Since its release in October 2006, Chronos has been used in more than 100 organizations for various purposes. It is routinely used for teaching of real-time systems in undergraduate courses in various universities such as University of British Columbia. The tool has also been augmented and used lately to analyze time bounds for system calls and interrupt latencies in a widely deployed operating system which has been widely used commercially in wireless devices (joint work with Gernot Heiser and his group at NICTA).

I have been an international invited member of the ArtistDesign Network of Excellence, the European Network of Excellence for embedded systems. As part of my work in ArtistDesign, I have played an active role on embedded software analysis for multi-core platforms. In this regard, Chronos was also extended recently (2012) to provide the first software timing analyzer for software running on multi-core platforms (joint work with Peter Marwedel and his group at T.U. Dortmund, Germany).

The papers [RTSS11-cache, RTSS13-cache, RTAS11] received best paper award nominations from IEEE Real-time Systems Symposium (RTSS) – the premier conference for real-time systems research. In terms of citations, our paper on Cache Related Pre-emption Delay analysis [CODES03] has 90 citations. The work on shared cache analysis [RTSS09-sharedcache] has 51 citations. The paper on Chronos tool [SCP] has 80+ citations. Also, the work [RTS06] is a full version of the earlier conference publication [RTSS04] on WCET analysis of processor pipelines. These works have 57+41 = 98 citations as of August 2013 (data from Google Scholar).

V. Future research plans

Programming is often taught as a stand-alone activity, but is seldom so in real-life. Programming in any organization in real-life typically involves making small but subtle changes in a large code-base. While code changes may be small, the semantic impact of such changes may be hard to gauge. In my work on semantic analysis based debugging, I have focused on building change-aware programming environments, and shown their scalability. In the longer term, it is essential to study not only the scalability but also the ease-of-use of such programming environments. As a first step in this direction, we have studied the possibility of providing a convenient contract language, called change contracts, for describing the intended behavior of program changes [ISSTA13]. We have studied the usability of such change contracts on human subjects. Once such change contracts are available, change-aware programming / analysis / testing and debugging become a possibility. Furthermore, large-scale industrial software development typically proceeds by checking-in of program versions where commit logs capture the changes being committed. There exists the possibility of extracting change contract specifications

automatically from commit logs, which aid the construction of change-aware programming environments and processes. Natural language processing techniques may be useful for such contract extraction.

One can also envision other key developments in software technologies in the next two decades. With the wide prevalence of software controlled embedded systems, their interconnectedness, and the growing trend towards “Internet of things” – non-functional properties of software such as timing, energy consumption and security are becoming as important as software functionality. In other words software development / testing / analysis / debugging has to consider several dimensions such as timing, energy and security. In the real-time embedded systems community, such a multi-dimensional view has long been studied, albeit at the system level. Thus, the construction of an embedded system design amounts to an exploration of the design points in a design space by optimizing multiple criteria such as timing, energy, cost and so on. This activity is commonly known as “design space exploration”. In future, one can envision that such a multi-criteria or multi-objective outlook will dominate software construction too. However, for software construction – we may not go through an explicit process of design space exploration. Instead, the programming activity may be made timing-aware, energy-aware, security-aware and so on. This involves systematic performance testing and debugging, energy testing and debugging and so on. While point technologies may exist for such purposes, a systematic methodology for testing and analysis of non-functional properties is missing. We have taken the first steps in this direction in our recent works [LCTES13, RTSS13-cache] in 2013, and I plan to develop this theme further in the future.

Concretely, we note that emerging computing platforms and environments, such as smart-phones / cloud services, demand the co-existence of critical and non-critical software whose functionality, timing, energy, security need to tested and guaranteed in a seamless fashion. Thus, critical and non-critical data / applications will continue to physically co-exist in the emerging computing platforms, and yet we need to impose a logical isolation between critical and non-critical application. Such a logical isolation can be achieved at different levels of granularity, and has wide commercial applications such as safe and secure mobile banking. Currently, I am in the process of coordinating a joint academia-industry research effort in this direction, which has received formal endorsement for sizeable industry funding for five years. We are discussing with funding agencies and other authorities on the possibility of taking forward the research effort from 2014-15.

REFERENCES

Software testing and analysis

[FSE09] DARWIN: An Approach for Debugging Evolving Programs
Dawei Qi, Abhik Roychoudhury, Zhenkai Liang, Kapil Vaswani, Joint meeting of ESEC and ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), ESEC-FSE 2009.

[FSE10] Golden Implementation Driven Software Debugging, Ansuman Banerjee, Abhik Roychoudhury, Johannes Harlie, Zhenkai Liang, ACM SIGSOFT Symp. on Foundations of Software Engineering (FSE) 2010.

[FSE11] Path Exploration based on Symbolic Output , Dawei Qi, Hoang D.T. Nguyen, *Abhik Roychoudhury*, Joint meeting of ESEC and ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), ESEC-FSE 2011.

[ICSE13-PRV] Partition-based Regression Verification, Marcel Bohme, Bruno C.d.S. Oliveira, Abhik Roychoudhury, ACM/IEEE International Conference on Software Engineering (ICSE) 2013.

[ICSE13-SEMFIX] SemFix: Program Repair via Semantic Analysis, Hoang D.T. Nguyen, Dawei Qi, Abhik Roychoudhury, Satish Chandra, ACM/IEEE Intl. Conference on Software Engineering (ICSE) 2013.

[ASE10] Test Generation to Expose Changes in Evolving Programs, Dawei Qi, Abhik Roychoudhury, Zhenkai Liang, 25nd IEEE/ACM Intl Conference on Automated Software Engineering (ASE) 2010.

[FSE13] Regression Tests to Expose Change Interaction Errors Marcel Böhme, Bruno C.d.S. Oliveira, Abhik Roychoudhury, ESEC/FSE '13, Joint meeting of ACM SIGSOFT symposium and European conference on Foundations of software engineering, 2013.

[ICSE04] Using Compressed Bytecode Traces for Slicing Java Programs, Tao Wang and Abhik Roychoudhury, ACM/IEEE International Conference on Software Engineering (ICSE) 2004.

[TOPLAS08] Dynamic Slicing on Java Bytecode Traces, Tao Wang and Abhik Roychoudhury ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 30(2), March 2008.

[ISSTA07] Hierarchical Dynamic Slicing , Tao Wang and Abhik Roychoudhury ACM International Symposium on Software Testing and Analysis (ISSTA) 2007.

[TOSEM09] Interacting Process Classes, Ankit Goel, Abhik Roychoudhury, and P.S. Thiagarajan, ACM Transactions on Software Engineering and Methodology (TOSEM), 18(4), 2009.

[FSE07, TOSEM12] Symbolic Message Sequence Charts, Abhik Roychoudhury, Ankit Goel and Bikram Sengupta, ACM Transactions on Software Engineering and Methodology (TOSEM), 21(2), 2012. Earlier version in ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE) 2007.

[ICSE11] Mining Message Sequence Graphs, Sandeep Kumar, Siau-Cheng Khoo, Abhik Roychoudhury, David Lo, ACM/IEEE International Conference on Software Engineering (ICSE) 2011.

[ICSE12] Inferring Class Level Specifications for Distributed Systems, Sandeep Kumar, Siau-Cheng Khoo, Abhik Roychoudhury and David Lo, ACM/IEEE International Conference on Software Engineering (ICSE) 2012.

[ISSTA13] Expressing and Checking Intended Changes via Software Change Contracts, Jooyong Yi, Dawei Qi, Shin Hwei Tan, Abhik Roychoudhury, International Symposium on Software Testing and Analysis (ISSTA) 2013.

Real –time embedded software

[SCP] Chronos: A Timing Analyzer for Embedded Software , Xianfeng Li, Yun Liang, Tulika Mitra and Abhik Roychoudhury, Science of Computer Programming, Volume 69, December 2007.

[RTSS04] Modeling Out-of-Order Processors for Software Timing Analysis, Xianfeng Li, Abhik Roychoudhury and Tulika Mitra, IEEE Real-Time Systems Symposium (RTSS) 2004.

[RTS05] Modeling Control Speculation for Timing Analysis, Xianfeng Li, Tulika Mitra and Abhik Roychoudhury, Real-Time Systems Journal, Kluwer Academic Publishers, 29(1), Jan 2005.

[RTS06] Modeling Out-of-Order Processors for WCET Analysis, Xianfeng Li, Abhik Roychoudhury and Tulika Mitra, Real-Time Systems Journal, Springer, 34(3), pages 195-227, 2006.

[RTSS09-ucache] Unified Cache Modeling for WCET Analysis and Layout Optimizations Sudipta Chattopadhyay and Abhik Roychoudhury, IEEE Real-time System Symposium (RTSS) 2009.

[CODES03] Accurate Estimation of Cache-related Preemption Delay, Hemendra Singh Negi, Tulika Mitra and Abhik Roychoudhury, ACM Intl. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2003.

[RTSS09-sharedcache] Timing Analysis of Concurrent Programs Running on Shared Cache Multi-cores Yan Li, Vivy Suhendra, Yun Liang, Tulika Mitra and Abhik Roychoudhury, IEEE Real-time System Symposium (RTSS) 2009.

[RTSJ-to-appear] Static Analysis of Multi-core TDMA Resource Arbitration Delays, Timon Kelter, Heiko Falk, Peter Marwedel, Sudipta Chattopadhyay and Abhik Roychoudhury, Real-time Systems Journal, To appear.

[RTAS12] A Unified WCET Analysis Framework for Multi-core Platforms , Sudipta Chattopadhyay, Chong Lee Kee, Abhik Roychoudhury, Timon Kelter, Peter Marwedel and Heiko Falk 18th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS) 2012.

[TOPLAS10] Scratchpad Allocation for Concurrent Embedded Software , Vivy Suhendra, Abhik Roychoudhury and Tulika Mitra, ACM Transactions on Programming Languages and Systems (TOPLAS), 32(4), April 2010.

[LCTES11] Static Bus Schedule aware Scratchpad Allocation in Multiprocessors, Sudipta Chattopadhyay and Abhik Roychoudhury, ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES) 2011.

[RTSS11-OS] Timing Analysis of a Protected Operating System Kernel, Bernard Blackham, Yao Shi, Sudipta Chattopadhyay, Abhik Roychoudhury and Gernot Heiser, IEEE Real-time Systems Symposium (RTSS) 2011.

[RTSS13-OS] Integrated Timing Analysis of Application and Operating Systems Code, Lee Kee Chong, Clement Ballabriga, Van-Thuan Pham, Sudipta Chattopadhyay, Abhik Roychoudhury, IEEE Real-time Systems Symposium (RTSS) 2013.

[RTSS11-cache] Scalable and Precise Refinement of Cache Timing Analysis via Model Checking Sudipta Chattopadhyay and Abhik Roychoudhury, IEEE Real-time Systems Symposium (RTSS) 2011.

[RTSS13-cache] Static Analysis driven Cache Performance Testing, Abhijeet Banerjee, Sudipta Chattopadhyay, Abhik Roychoudhury, IEEE Real-time Systems Symposium (RTSS) 2013.

[RTS12] Performance Debugging of Esterel Specifications, Lei Ju, Bach Khoa Huynh, Abhik Roychoudhury and Samarjit Chakraborty, Real-time Systems Journal, 48(5), 2012.

[CODES08] Cache-aware Optimization of BAN Applications Yun Liang, Lei Ju, Samarjit Chakraborty, Tulika Mitra and Abhik Roychoudhury, ACM Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2008.

[RTAS11] Scope-aware Data Cache Analysis for WCET Estimation, Bach Khoa Huynh, Lei Ju and Abhik Roychoudhury, 17th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS) 2011.

[LCTES13] Program Performance Spectrum, Sudipta Chattopadhyay, Lee Kee Chong, Abhik Roychoudhury, ACM SIGPLAN Conference on Languages, Compilers and Tools for Embedded Systems (LCTES) 2013.
[RTSS09-sharedcache] Timing Analysis of Concurrent Programs Running on Shared Cache Multi-cores Yan Li, Vivy Suhendra, Yun Liang, Tulika Mitra and Abhik Roychoudhury, IEEE Real-time System Symposium (RTSS) 2009.

[RTSJ-to-appear] Static Analysis of Multi-core TDMA Resource Arbitration Delays, Timon Kelter, Heiko Falk, Peter Marwedel, Sudipta Chattopadhyay and Abhik Roychoudhury, Real-time Systems Journal, To appear.

[RTAS12] A Unified WCET Analysis Framework for Multi-core Platforms , Sudipta Chattopadhyay, Chong Lee Kee, Abhik Roychoudhury, Timon Kelter, Peter Marwedel and Heiko Falk 18th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS) 2012.

[TOPLAS10] Scratchpad Allocation for Concurrent Embedded Software , Vivy Suhendra, Abhik Roychoudhury and Tulika Mitra, ACM Transactions on Programming Languages and Systems (TOPLAS), 32(4), April 2010.

[LCTES11] Static Bus Schedule aware Scratchpad Allocation in Multiprocessors, Sudipta Chattopadhyay and Abhik Roychoudhury, ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES) 2011.

[RTSS11-OS] Timing Analysis of a Protected Operating System Kernel, Bernard Blackham, Yao Shi, Sudipta Chattopadhyay, Abhik Roychoudhury and Gernot Heiser, IEEE Real-time Systems Symposium (RTSS) 2011.

[RTSS13-OS] Integrated Timing Analysis of Application and Operating Systems Code, Lee Kee Chong, Clement Ballabriga, Van-Thuan Pham, Sudipta Chattopadhyay, Abhik Roychoudhury, IEEE Real-time Systems Symposium (RTSS) 2013.

[RTSS11-cache] Scalable and Precise Refinement of Cache Timing Analysis via Model Checking Sudipta Chattopadhyay and Abhik Roychoudhury, IEEE Real-time Systems Symposium (RTSS) 2011.

[RTSS13-cache] Static Analysis driven Cache Performance Testing, Abhijeet Banerjee, Sudipta Chattopadhyay, Abhik Roychoudhury, IEEE Real-time Systems Symposium (RTSS) 2013.

[RTS12] Performance Debugging of Esterel Specifications, Lei Ju, Bach Khoa Huynh, Abhik Roychoudhury and Samarjit Chakraborty, Real-time Systems Journal, 48(5), 2012.

[CODES08] Cache-aware Optimization of BAN Applications Yun Liang, Lei Ju, Samarjit Chakraborty, Tulika Mitra and Abhik Roychoudhury, ACM Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2008.

[RTAS11] Scope-aware Data Cache Analysis for WCET Estimation, Bach Khoa Huynh, Lei Ju and Abhik Roychoudhury, 17th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS) 2011.

[LCTES13] Program Performance Spectrum, Sudipta Chattopadhyay, Lee Kee Chong, Abhik Roychoudhury, ACM SIGPLAN Conference on Languages, Compilers and Tools for Embedded Systems (LCTES) 2013.

Selected Publications of Abhik Roychoudhury

Email: abhik@comp.nus.edu.sg
http://www.comp.nus.edu.sg/~abhik

PUBLICATION summary

The full publication list from NUS Staff research publication system appears at the end of the research section of this dossier, and hence is not repeated here. University/school also maintains a ranking of journals and conferences into tiers, and the data in the following contains a summary of my publications with respect to this ranking. *I have co-authored 46 tier-1 conference and journal papers, after my PhD*. All these papers were co-authored during my employment at NUS – first as Assistant Professor, and then as Associate Professor.

Break-up into areas My two main areas of research are Software Engineering and Real-time Systems. In both of these areas I have published regularly in the top venues. As evidence, I can mention at least 20 papers co-authored after my PhD in premier (tier-1) conferences and journals in Real-time / Embedded Systems, and at least another 26 papers co-authored after my PhD in premier (tier-1) conferences and journals in Software Engineering/Programming Languages.

Conference vs. Journal As is well known, tier-1 conferences in Computer Science are extremely prestigious, competitive and peer-reviewed. I have co-authored 31 publications in premier (tier-1) conferences in my areas of research. Over and above the conference publications, I have co-authored 15 journal papers in premier (tier-1) journals, after my PhD.

STATEMENT on CO-AUTHORSHIP

All my co-authored publications involve substantial contribution from me, and my co-authors.

Whenever a student is involved, the student's name appears ahead of the faculty members – irrespective of the student's contribution. This is done with the broader goal of giving greater visibility to student authors early in their career. In other words, the names of faculty members appear after student names.

If several faculty members are involved in a paper – among the faculty members the names are typically arranged in alphabetical order of surnames. The alphabetical order of surnames among faculty members is reversed only in rare cases, when the contribution of a faculty member is significantly higher than the other faculty member.

SELECTED PUBLICATIONS

Following is a list of selected publications co-authored by me. For each publication, the significance of the publication is also explained. Among the chosen publications, there is no particular order. I have listed them chronologically – ICSE 2004, SCP 2007, ESEC-FSE 2009, ESEC-FSE 2011, RTSS 2013.

[ICSE04] *Using Compressed Bytecode Traces for Slicing Java Programs*
Tao Wang and Abhik Roychoudhury, ACM/IEEE International Conference on Software Engineering (ICSE) 2004.

[Journal paper with the title "Dynamic Slicing on Java Bytecode Traces" appearing in ACM Transactions on Programming Languages and Systems (TOPLAS), 30(2), 2008.]

Significance of the work: Current software debugging tools require active participation from the programmer to find the cause(s) of an observable error. This makes software debugging tedious and extremely time-consuming. It is well-known that developers routinely validate their programs by testing. However, if the output of a test case is unexpected, there does not exist suitable tools to automatically analyze the failed program execution for possible error causes. In this work [ICSE04], we have developed the first dynamic slicing tool JSlice for a full-scale programming language. Given an observable error found by program testing, it can perform automated program dependence analysis to highlight the probable error causes. Apart from showing the scalability of dynamic slicing, and its applicability to a widely used programming language, this work also makes an important conceptual contribution. It shows that the analysis and dependency extraction needed for dynamic slicing can be achieved in the "compression domain", without decompressing the trace representation. Indeed, this is a key factor contributing to the scalability of our dynamic slicing method. The JSlice tool reported in [ICSE04,TOPLAS08] has been used in over 300 organizations in 30 countries for teaching, research and development. The work [TOPLAS08] is a full version in a journal of the preliminary conference paper [ICSE04]. These works have 60+28 = 88 citations in total, as of August 2013 (data from Google scholar).

[SCP] *Chronos: A Timing Analyzer for Embedded Software*
Xianfeng Li, Yun Liang, Tulika Mitra and Abhik Roychoudhury
Science of Computer Programming, Volume 69, December 2007.

Significance of the work: Timing is critical to the functioning of many computing systems. Indeed, real-time embedded systems are ubiquitous and control the functionality of many devices used in our everyday lives. Even safety critical application domains, such as modern day cars, are largely controlled by software. Many of such software is time-critical, that is, the execution time of the software is critical to its correct functioning. One primary example along such lines is the software controlling the brakes of a modern car. In this line of work, our aim is to develop methods which can estimate the Worst-case Execution Time (WCET) estimate of a given software – an upper bound of the execution time it can take for any possible input. Such an estimate should be safe (it should not underestimate the possible execution times) as well as tight (it should not greatly overestimate the actual execution times). WCET estimation methods have been studied for the past two decades and various methods / tools have been developed. The challenges we tackle are one of scalability – analyzing large-scale embedded software running on new generation environments/ platforms. Our main effort in this direction is embodied by

Chronos [SCP], a worst-case execution time analysis tool for C binaries. Chronos is an execution time analysis tool which can statically analyze C programs to predict safe and tight time bounds. The tool employs many technical novelties including the modeling of timing effects of many novel micro-architectural features in the underlying processor platform. Since its release in October 2006, Chronos has been used in more than 150 organizations for various purposes. It is routinely used for teaching of real-time systems in undergraduate courses in various universities such as University of British Columbia. The tool has also been augmented and used lately to analyze time bounds for system calls and interrupt latencies in a widely deployed operating system which has been widely used commercially in wireless devices (joint work with Gernot Heiser and his group at NICTA). The paper [SCP] has 80+ citations (as of August 2013, data from Google Scholar), and the Chronos tool has several other citations as can be established by searching for www.comp.nus.edu.sg/~rpembed/chronos from Google scholar.

Furthermore, we have built on top of Chronos with timing models of shared cache and shared bus, as evidenced in our later works e.g. Timing Analysis of Concurrent Programs Running on Shared Cache Multi-cores, RTSS 2009 (~50 citations in Google Scholar). These timing models of multi-core architectures have also been combined with our accurate timing model of a single core (capturing caches, pipeline and branch prediction). This has led a unified framework for predicting the execution time of software running on multi-core platforms http://www.comp.nus.edu.sg/~rpembed/chronos-multi-core.html as described by our paper in RTAS 2012.

[FSE09]

DARWIN: An Approach for Debugging Evolving Programs Dawei Qi, Abhik Roychoudhury, Zhenkai Liang, Kapil Vaswani, Joint meeting of ESEC and ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), ESEC-FSE 2009.

[Journal paper with same title appearing in ACM Transactions on Software Engineering and Methodology (TOSEM), 21(3), 2012.]

Significance of the work: Large software projects typically do not start from scratch. Instead, over time a piece of software evolves with possibly more features being added, and more optimizations being integrated. In trying to add more functionality, programmers often inadvertently break existing functionality – this is commonly known as a *regression*. Finding the root-cause of such regressions (or debugging) is a major head-ache for software developers in *any* large development project! The inherent difficulty in debugging lies in the lack of capture of intended software behavior. If the intended behavior is available in the form of a formal specification, one can locate the deviation of the actual observed behavior from the intended behavior, and thereby locate the root-cause. However, in real-life, programmers are reluctant to put in the extra effort for writing formal specifications. Since programming hardly occurs from scratch, we propose to use the previous working version of the program, as an informal specification of its intended behavior. In [FSE09], we use symbolic execution of past and current program versions to localize the root cause of software regressions. Symbolic execution corresponds to executing a program with symbolic or un-instantiated inputs, as opposed to concrete input values. This produces logical formulae which serve as a semantic footprint of the program execution. The paper “DARWIN: an approach for debugging evolving programs” [FSE09] received the ACM SIGSOFT Distinguished paper Award, and was also show-cased at the Microsoft Professional Developer’s Conference (PDC) in 2009. This work has ~40

citations as of August 2013 (data from Google Scholar). The DARWIN work is now credited for being the first work in "formula based debugging" where logical formula are used describe program executions, and these logical formulae are analyzed for debugging. Subsequently researchers in various places such as IBM, NYU, Max Planck and other places have started investigating in this area, as shown by the subsequent papers in the area. I have been invited to invitation-only events to share my perspectives on the use of symbolic techniques for software debugging. These include the Dagstuhl Seminar on Fault Prediction, Localization and Repair – Germany (February 2013) and PAS 2012 International Seminar on Program Verification, Automated Debugging and Symbolic Computation - Beijing, China, organized by Chinese Academy of Sciences (October 2012).

[FSE11] *Path Exploration based on Symbolic Output* Dawei Qi, Hoang D.T. Nguyen, Abhik Roychoudhury Joint meeting of ESEC and ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), ESEC-FSE 2011.

[Journal paper with same title to appear in ACM Transactions on Software Engineering and Methodology (TOSEM).]

Significance of the work: Software testing can be seen as an attempt to gain confidence in the program behavior. In this context, testing based on coverage of various program artifacts have been studied – including statement coverage, branch coverage, path coverage. Most recently symbolic execution based testing has gained prominence, which tries to achieve path coverage. Efficient program path exploration is important for many software engineering activities such as testing, debugging and verification. Why do we attempt to cover more paths in software testing? The implicit assumption here is that by covering more paths, we are likely to cover more of the possible behaviors that can be exhibited by a program. However, as is well known, path enumeration is extremely expensive. In this paper, we develop a partitioning of program paths based on the program output. Two program paths are placed in the same partition if they derive the output similarly, that is, the symbolic expression connecting the output with the inputs is the same in both paths. Our partitioning of paths is created on-the-fly via a smart path exploration. Most interestingly, we show that the path partitions can be effectively computed on-the-fly via a smart dependency analysis. We show that starting from a random input, extensions on dynamic slicing on the trace of the input can be used to compute partitions of the input domain --- if any two inputs share the same symbolic output, their extended slices must be the same. We also show that an on-the-fly search based on the extended slices is guaranteed to be complete, that is, we explore all the partitions in the input domain.

The immediate applications of the work are in software testing, where instead of achieving path coverage, we can more efficiently achieve path partition coverage. Apart from the application of the work in software testing, I feel that the technical results and the key theorems in the work are valuable in their own right. I believe that the paper presents a hard technical result in software testing and analysis, which helps us understand important characteristics of transformational program behavior – namely how the computation along program paths can be neatly grouped together based on a combination of static and dynamic program dependencies.

[RTSS13]
Static Analysis Driven Cache Performance Testing
Abhijeet Banerjee, Sudipta Chattopadhyay, Abhik Roychoudhury,
IEEE Real-time Systems Symposium (RTSS) 2013.

Significance of the work:
With the increasing deployment of software in real-time embedded platforms, testing of non-functional properties of software has become extremely crucial. Real-time, embedded software are required to satisfy several extra-functional properties, such as timing. Therefore, performance validation marks a crucial stage before certifying such time-critical software. In the absence of appropriate performance validation techniques, the deployed software may suffer from severe performance problems, such as missing deadlines. Conventionally performance testing of software has been conducted on an ad-hoc basis. In this work, we focus on performance degradation due to cache thrashing. With the ever increasing performance gap between the processor and the main memory, the performance of memory subsystems often pose a significant bottleneck in achieving the desired performance for real-time, embedded software. Cache memory plays a key role in reducing the performance gap between a processor and main memory. Therefore, analysing the cache behaviour of a program is critical for validating the performance of embedded software. The main observation behind our approach is to convert performance testing problem into a problem which can be solved by path-based functionality testing. We first carry out static cache analysis on the program to decide the set of program points that *may* exhibit cache thrashing. Subsequently, we systematically generate assertions at such places to expose cache thrashing in the program itself. The required functionality of the software is augmented with the set of assertions introduced by us. To check the validity of different assertions, we build a *dynamic path exploration* strategy that directs the path searching process towards the set of instrumented assertions. Since we dynamically explore the set of assertions, our computed test-suite does not contain any *false positives*. Precisely, any test case included in the computed test-suite captures a cache performance issue (specifically, a cache thrashing scenario) in some *feasible execution* of the software.

In a broader view, therefore, we reduce the problem of testing cache performance to an equivalent functionality testing problem. The test suite constructed provides coverage of all cache thrashing scenarios. This work is one of the first to take a formal systematic approach to performance testing of software. This is the newest among the chosen publications, and has been *nominated for best paper award* in RTSS 2013.

Research Performance Indicators of Abhik Roychoudhury

Email: abhik@comp.nus.edu.sg
http://www.comp.nus.edu.sg/~abhik

I. Publications in top venues during my employment at NUS

The full publication list from NUS Staff research publication system appears at the end of the research section of this dossier, and hence is not repeated here. University/school also maintains a ranking of journals and conferences into tiers, and the data in the following contains a summary of my publications with respect to this ranking. In total, I have published **46 papers in tier-1 conferences and journals**, after my PhD, and during my employment at NUS.

Break-up into areas My two main areas of research are Software Engineering and Real-time Systems. In both of these areas I have published regularly in the top venues. As an evidence of this, I can mention at least 20 papers co-authored after my PhD in premier (tier-1) conferences and journals in Real-time / Embedded Systems, and at least another 26 papers co-authored after my PhD in premier (tier-1) conferences and journals in Software Engineering / Programming Languages.

Conference vs. Journal As is well known, tier-1 conferences in Computer Science are extremely prestigious, competitive and peer-reviewed. I have co-authored 31 publications in premier (tier-1) conferences in my areas of research, after my PhD. Over and above the conference publications, I have co-authored 15 journal papers in premier (tier-1) journals, after my PhD.

Flagship venues in Software Engineering I have co-authored 7 full-length papers in the main technical track of International Conference on Software Engineering (ICSE), the flagship venue in Software Engineering. In addition, I have co-authored 5 papers in ACM SIGSOFT (also called FSE or ESEC-FSE), and 4 papers in ACM Transactions on Software Engineering and Methodology (TOSEM).

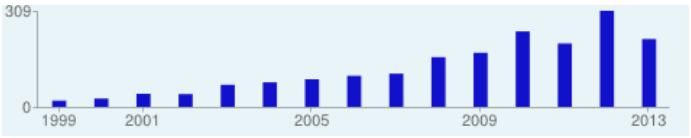
Flagship venues in Real-time Systems I have co-authored 9 full-length research papers in the IEEE Real-time Systems Symposium (RTSS), the flagship conference in real-time systems. I have also co-authored 7 papers in Real-time systems Journal, the flagship journal in real-time systems research.

II. Citation and Impact Analysis

a) Overall citations

Citation indices

	All	Since 2008
Citations	1936	1313
h-index	26	20
i10-index	53	43



Citations to my articles

In the preceding, I have provided the summary of citations – as collected from Google scholar on 11th September 2013. These data are updated by Google scholar regularly.

- h-index is the largest number h, such that h publications have h citations.
- i-10 index is the number of publications with more than 10 citations.
- For my publications, both h-index and i-10 index are relatively high – indicating that a large number of my papers have attracted attention.
- In total, I have co-authored 9 papers with more than 50 citations – data from Google Scholar, as of September 11, 2013.

b) Positioning of my citations in the areas I work in

Different areas in Computer Science have different volumes of citations. So, apart from reporting the total number of citations, I can also point to my relative position in terms of citations in the topics/areas I work on. In my Google Scholar author profile, these areas are listed as

Software Testing - Program Analysis - Real-time Systems

By looking up these areas in Google Scholar, one can see my relative position in terms of citations among researchers working on Software testing, Program Analysis, Real-time Systems respectively. As of September 11 2013, my relative positions are as follows.

- #32 among researchers in Software Testing
- #28 among researchers in Program Analysis
- #29 among researchers in Real-time Systems.

These data are updated dynamically, so these positions may change over months, but not very significantly.

c) Papers with highest citation counts

Some of my papers with higher number of citations are stated in the following.

[90 citations] Accurate Estimation of Cache-related Preemption Delay
ACM Intl. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2003.

[84 citations] WCET centric data allocation to scratchpad memory
IEEE Real-Time Systems Symposium (RTSS) 2005.

[81 citations] Chronos: A Timing Analyzer for Embedded Software
Science of Computer Programming, Volume 69, December 2007.

[78 citations] XMC: A logic programming based verification toolset
Computer Aided Verification (CAV) 2000.

[74 citations] Using formal techniques to Debug the AMBA System-on-Chip Bus Protocol
IEEE/ACM Conference on Design Automation and Test in Europe (DATE) 2003.

In addition, I can also mention the following two papers, which have each appeared in a conference first, and then an augmented version has appeared in a journal.

[ICSE04] Using Compressed Bytecode Traces for Slicing Java Programs,
Tao Wang and Abhik Roychoudhury
ACM/IEEE International Conference on Software Engineering (ICSE) 2004.

[TOPLAS08] Dynamic Slicing on Java Bytecode Traces
Tao Wang and Abhik Roychoudhury
ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 30(2), March 2008.

ICSE2004 paper has 60 citations plus journal paper in TOPLAS has 28 citations, leading to a total of 60+28 = 88 citations.

Modeling Out-of-Order Processors for Software Timing Analysis
Xianfeng Li, Abhik Roychoudhury and Tulika Mitra
IEEE Real-Time Systems Symposium (RTSS) 2004.

Modeling Out-of-Order Processors for WCET Analysis
Xianfeng Li, Abhik Roychoudhury and Tulika Mitra
Real-Time Systems Journal, Springer, 34(3), pages 195-227, 2006.
A much expanded version of our RTSS 2004 paper on the same topic.

The journal paper has 57 citations, plus the RTSS 2004 paper has 41 citations, leading to a total of 57 + 41 = 98 citations.

d) Analysis of Research Impact that is not captured by citations

My research on program analysis and embedded software has produced analysis tools which are widely used for teaching, research and other purposes at universities, companies and other organizations. In this regard, I would highlight the following tools that have resulted from my research, and are widely used.

- The JSlice tool is a dynamic Slicing for Java program debugging and comprehension. It resulted from my work in [ICSE04,TOPLAS08]. Released in 2006-07, it has users in over 300 organizations in many different countries. The user list of the tool is also attached in the "Research Portfolio" part in the supplementary dossier.

- The Chronos tool is a Worst-case Execution Time (WCET) analysis tool for C programs. It resulted from my research [SCP07]. Released in 2006, it has users in over 150 organizations (universities & companies), and is used in under-graduate teaching in universities outside NUS, such as the University of British Columbia. It was also a successful participant in the first WCET Tool Challenge. The user list of Chronos is also attached in the "Research Portfolio" part in the supplementary dossier.

In addition, my research has always had substantial connections to the industry leading to industrial impact. In its most visible form, we have currently received a *formal endorsement for a substantial project on High Assurance Software from an industrial consortium of multi-national software companies. The funding commitment is for five million dollars over a five year period.*

III. Research Awards and Honors

Following are some selected research awards and honors that I have received.

- ***ACM Distinguished Speaker*** *– appointed May 2013.* The Association for Computing Machinery (ACM), the world's largest educational and scientific computing society, " provides colleges and universities, corporations, event and conference planners, and agencies - in addition to ACM local Chapters - with direct access to top technology leaders and innovators from nearly every sector of the computing industry." The ACM Distinguished Speaker program features "renowned thought leaders in academia, industry and government, speaking about the most important topics in computing and IT world." The DSP committee "votes on the acceptability of a nominated speaker. Each DSP Committee member will rate the nominee on a 1-to-5 scale; this is the rating scale used by ACM's Fellows Committee". More details about the program is available from http://dsp.acm.org/

- *ACM SIGSOFT Distinguished Paper Award, given in SIGSOFT FSE 2009 for the paper "DARWIN: An approach for Debugging Evolving Programs".* The ACM Special Interest Group on Software Engineering (SIGSOFT) provides a forum for computing professionals from industry, government and academia to examine principles, practices, and new research results in software engineering. ACM SIGSOFT encourages SIGSOFT-sponsored conferences to designate a number of accepted papers (up to 10%) for ACM SIGSOFT Distinguished Paper Awards for the conference. In FSE 2009, 3 out of 250 submitted papers received the award. FSE is one of the two flagship conferences in Software Engineering.

- *IBM Faculty Award, Awarded in Dec 2008 / Jan 2009.* This award is given by IBM to selected faculty members worldwide. The award fund can be used in any way for research, without monitoring from IBM. The prize is typically awarded to researchers whose research is of relevance to IBM. I was the first faculty member in NUS School of Computing (and possibly the second in NUS) to receive this award. I believe this shows the relevance of my research for computing industry as well.

- *Tan Kah Kee Young Inventor Award (Silver) 2008* This was a Young Inventor Award which was given for my research on the JSlice tool for software debugging and comprehension.

- *Best paper award nominations in many top-tier conferences including RTSS 2011, RTSS 2013.*

IV. Research grants – external research funding obtained

In the following, I mention only the external research grants. *Grants at Faculty Research Council (FRC) or University Research Council (URC) level are **not** mentioned.* First, I list the external grants in which I am PI, followed by the grants in which I am Co-PI. All grant amounts are in SGD, unless otherwise stated.

a) External grants obtained as Principal Investigator (PI) – most recent first.

As can be seen from the following listing, currently I have four active grants from different funding agencies. I am the PI in all of these grants, with no Co-PIs involved. The total amount of grant funding from these four active grants is over **$2.1 million**.

Project Name	Funding Agency	Amount	Duration	Start date	My role
Energy Aware Programming	MoE Tier 2	$373K	3 years	Jan 2014 [*active*]	PI [no Co-PIs]
CoDeTest: Comprehension Detection and Testing via Symbolic Execution	DSO Labs	$390K	2 years	June 2013 [*active*]	PI [no Co-PIs]
Scalable Timing Analysis Methods for Embedded Software	A*STAR Public Sector Funding (PSF)	$590K	3 years	Feb 2012 [*active*]	PI [no Co-PIs]
Analysis and Test Generation for Evolving Software	MoE Tier 2	$830K	3 years	May 2011 [*active*]	PI [no Co-PIs]
Symbolic Taint Analysis	DSTA Defense Innovative Research Program (DIRP)	$397K	3 years	Feb 2009 [concluded]	PI [Co-PI: Liang Zhenkai]
Correctness and Performance Issues in the CLI Memory Model	Microsoft	USD 15K	1 year	July 2005 [concluded]	PI [Co-PI: P.S. Thiagarajan]
Tools and techniques for Model based Software Debugging	A*STAR Public Sector Funding (PSF)	$362K	3 years	Sept 2004 [concluded]	PI [Co-PIs: None]

b) External grants obtained as Co-Principal Investigator (Co-PI) – most recent first.

Project name	Funding Agency	Amount	Duration	Start Date
EASEL: Engineering Architectures and Software for the Embedded Landscape	A*STAR	$1.35 million	3 years	March 2006 [concluded]
Formal Design Techniques for Reactive Embedded Systems	A*STAR	$429K	3 years	March 2003 [concluded]
Reactive Embedded Systems: High level Design Methods	A*STAR – Pilot project	$29K	1 year	Nov 2001 [concluded]

V. Membership of international boards and networks

[2009 -12]
International Member of ArtistDesign Network of Excellence on Design for Embedded Systems, this is the European Network of Excellence in Embedded Systems. My technical roles were related to the cluster on Software Synthesis, Code Generation and Timing Analysis.

[2012-15]
International Partner in COST Action on Timing Analysis on Code Level (TACLe).

VI. Chairmanship / membership of conference Committees

I have been appointed to conference organization of major conferences, including the following.

- Co-chair of New Ideas and Emerging Results (NIER) Track, International Conference on Software Engineering (ICSE) 2015.
- Co-chair of Mentoring Track, International Conference on Software Engineering (ICSE) 2014.
- Co-chair of Doctoral Symposium, ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE) 2014.
- Chair of Design and Verification Track, IEEE Real-time Systems Symposium (RTSS) 2012
- Program Chair, International Colloquium on Theoretical Aspects of Computing (ICTAC) 2012.

Apart from chairing the organization of conferences/tracks, I have been a member of Program Committee of at least fifty (50) conferences and workshops by now. The conferences where I have recently been a member of the Program Committee include the following (only a few recent ones are mentioned).

- RTSS – IEEE Real-time Systems Symposium – 2010, 2011, 2013.
- RTAS – IEEE Real-time Applications Symposium – 2014
- DATE – Design Automation and Test in Europe – 2012, 2014.
- FSE – ACM SIGSOFT Foundations of Software Engineering – 2012
- ISSTA – International Symposium on Software Testing and Analysis – 2013
- ASE – Automated Software Engineering – 2013.
- ATVA – Automated Technology for Verification and Analysis – 2012, 2013.
- LCTES – ACM SIGPLAN Symposium on Languages Compilers and Tools for Embedded Systems – 2012.
- ICST – International Conference on Software Testing – 2013, 2014.
- ICSE – International Conference on Software Engineering – 2014(NIER), 2009(Tools).
- Panelist in ISSTA 2013 Doctoral Symposium

VII. Invited presentations at scientific meetings / workshops

Following are some selected invited presentations I have delivered. Some of these are talks at invitation-only events, and the rest are keynote style talks or technical briefings. Where the same talk is given on several occasions, they are clubbed together as a single entry.

[Invitation-only event] How Symbolic Reasoning can Help Program Debugging and Repair**,** International Workshop on Future of Debugging, July 15 2013, Lugano, Switzerland, Co-located with International Symposium on Testing and Analysis (ISSTA) 2013.

[Invitation-only-event] How Timing Debugging can Benefit from Functionality Debugging, Workshop on Multi-core Application Debugging, Industry-Academia joint event, Germany, November 2013.

[Invitation-only event] SEMFIX: Program Repair via Semantic Analysis, Dagstuhl Seminar on Fault Prediction, Localization and Repair, Dagstuhl, Germany, February 2013.

[Invitation-only event] Formal techniques for debugging software regressions, Invited talk at International Seminar on Program Debugging, Automated Verification and Symbolic Computation (PAS) 2012, Organized by Chinese Academy of Sciences and Beihang University, Beijing, China, October 2012.

[Keynote/Invited] Debugging as a Science, that too, for Evolving Programs, Keynote given at 3rd International Workshop on Harnessing Theories for Tool Support in Software (TTSS) 2009, a workshop held along with the International Colloqium on Theoretical Aspects of Computing (ICTAC) 2009.

[Keynote/Invited] Program Transformations for Automated Verification, International Conference on Logic Programming 2002, Copenhagen, Denmark.

[Technical Briefing] Symbolic Techniques for Software Debugging and Repair, ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC-FSE) 2013, Tutorial given jointly with Satish Chandra. A similar Technical Briefing was given at the Intl. Conf. on Software Engineering (ICSE) 2012.

[Invitation-only event] Interacting Process Classes, Talk given on several occasions such as Workshop on Formal Methods for Design and Analysis of Software organized by Microsoft Research India (2005), and second Workshop on Predictable software component Assembly organized by University of Manchester (2005).

VIII. **Service as Reviewer**

Reviewer of papers in many international journals, including

Formal Aspects of Computing Journal (FACJ), Fundamenta Informaticae (FI), Formal Methods in System Design (FMSD), International Journal on Foundations of Computer Science (IJFCS), Journal of Software and System Modeling (SoSyM), ACM Transactions on Architecture and Code Optimization (TACO), Theoretical Computer Science (TCS), ACM Transactions on Embedded Computing Systems (TECS), ACM Transactions on Programming Languages and Systems (TOPLAS), ACM Transactions on Software Engineering and Methodology (TOSEM), Theory and Practice of Logic Programming (TPLP), IEEE Transactions on Software Engineering (TSE)

and conferences, including

ACSD, APAQS, APSEC, ASIAN, ASP-DAC, CASES, CAV, CONCUR, CP, ECRTS, EMSoft, FM, FoSSaCs, FST&TCS, GPCE, ICALP, ICLP, ISoLA, LCTES, PADL, PEPM, POPL, PLDI, RTSS, VMCAI.

Placing my research profile externally – articulated by Abhik Roychoudhury

Email: abhik@comp.nus.edu.sg
http://www.comp.nus.edu.sg/~abhik

For the completeness of the dossier, I provide the positioning of my research interests within broader research communities.

I. Placing my research within Computer Science

My research in software engineering is primarily in testing and analysis. Currently, research on testing and analysis constitutes a bulk of submissions or accepted papers in major software engineering venues. As evidence of this, I note that in the recently concluded ACM SIGSOFT 2013 conference (also called ESEC-FSE 2013) – testing and analysis accounted for 21 out of the 51 accepted papers. Thus, 40% of the accepted papers in the ACM SIGSOFT 2013 was from testing and analysis – showing that it is a distinct and well-recognized sub-area within the software engineering community. These papers covered software testing, static and dynamic analysis, and software verification methods to support such analyses. One can also mention that the latest International Symposium on Software Testing and Analysis (ISSTA) 2013 attracted 124 submissions, out of which only 25% were accepted. This conference covers software testing, analysis and validation – the topics I research in.

My research in real-time systems focuses on formal analysis of real-time embedded software. This is a well-recognized sub-area within the real-time systems community, as shown by the fact that IEEE Real-time Systems Symposium (RTSS) regularly allocates a separate "*Design and Verification*" track for papers on these topics. RTSS is the premier and flagship conference in real-time systems.

II. Benchmarking against colleagues in other institutions

One of the considerations in putting up my research profile is to benchmark it with external colleagues in my broad research area, specifically those who have been promoted to full professors. We consider three separate dimensions – notable achievement, citations, and publications in flagship venues.

A. Based on a notable achievement

It is possible, at least to some extent, to look at the list of academics in a certain broad area – who have achieved a certain notable achievement. For this purpose, I provide the list of academics who have listed their interests as Software Engineering, and are ACM Distinguished Speaker, a distinction that I have achieved. These people are listed in the following. One can verify this list from http://dsp.acm.org/

- Margaret Burnett, Professor, Oregon State University, USA
- Laura Dillon, Professor, Michigan State University, USA
- Philippe Kruchten, Professor, University of British Columbia (UBC), Canada
- Leon Osterweil, Professor, University of Massachusetts Amherst, USA
- Hanan Samet, Professor, University of Maryland College Park, USA
- Sandeep Shukla, Professor, Virginia Tech, USA
- Jeanette Wing, Professor, CMU, USA
- Alexandar Wolf, Professor, Imperial College, UK

- Tao Xie, Associate Professor, University of Illinois at Urbana Champaign (UIUC), USA.
- *Abhik Roychoudhury, Associate Professor, National University of Singapore, Singapore*

As shown in the web-site, currently there are only 123 ACM Distinguished Speakers all over the world.

B. Benchmarking based on Citations:

Based on my research areas – I provide a comparison of my citation data with external colleagues. My citation data can be studied along with the data of the following external colleagues (two from Software Engineering and one from Real-time Systems), whose research areas are somewhat similar, and whose PhD time-line is not too far from mine. All of them have been full professors for some years, but I do not have the information about exactly when they became full professors.

Marsha Chechik, Professor, Department of Computer Science at University of Toronto, Canada.
PhD 1996 (has spent 17 years after PhD)
Researching in Software Engineering and Software Verification
Total citations = 2538, h-index = 28, i10-index = 65,
Growth of citations per year, in recent years = 250 – 360.

Dirk Beyer, Professor of Computer Science, University of Passau, Germany
PhD 2002 (has spent 11 years after PhD)
Researching in Program Analysis, Software Verification, and related topics
Total citations = 2231, h-index = 26, i10-index = 42
Growth of citations per year, in recent years = 250 -370.

Steve Goddard,
Professor and Chair, Department of Computer Science, University of Nebraska Lincoln, USA
PhD 1998 (has spent 15 years after PhD)
Researching in Real-time Systems, Embedded Systems and related topics
Total citations = 1903, h-index = 21, i10-index = 44
Growth of citations per year, in recent years = 200 - 250

Based on my Google scholar profile – following are my own data.
PhD in 2000 (I have spent 13 years after PhD)
Researching in software testing, and formal analysis, with focus on embedded software
Total citations = 1936, h-index = 26, i10 index = 53,
Growth of citations each year in recent years = 250-310.

All of the citation data were captured on September 11th 2013, from Google Scholar. h-index is the maximum value of h, such that h papers have at least h citations. i10 index is the number of papers with at least ten citations.

We note that different areas in Computer Science have different volumes of citations. So, apart from reporting the total number of citations, I can also point to my relative position in terms of citations in the topics/areas I work on. In my Google Scholar author profile, these areas are listed as

Software Testing - Program Analysis - Real-time Systems

By looking up these areas in Google Scholar, one can see my relative position in terms of citations among researchers working on Software testing, Program Analysis, Real-time Systems respectively. As of September 11 2013, my relative positions were as follows.

- #32 among researchers in Software Testing
- #28 among researchers in Program Analysis
- #29 among researchers in Real-time Systems.

These data are updated dynamically, so these positions may change over months, but not very significantly. Needless to say, there might be many other ways of benchmarking colleagues, apart from citations, such as awards or honors received – as was shown in my benchmarking based on colleagues who are ACM Distinguished Speakers.

C. Based on Publications in Flagship venues

To date, I have published 9 papers in RTSS, the premier conference in real-time systems. This includes 2 papers in RTSS 2013. However, the list of all accepted papers for RTSS 2013 is still not available, and the conference will take place in December 2013 – so I am considering only the number of RTSS papers published up to 2012. Up to 2012, I have published 7 papers in RTSS.

I have also published 7 papers in ICSE, the premier conference in software engineering. I compare these data, with the publication profile of the afore-mentioned external colleagues – Marsha Chechik and Dirk Beyer from Software Engineering, as well as Steve Goddard from Real-time embedded systems. All of the publication data is extracted from the well-known DBLP Bibliography server, and can be verified.

Name of researcher	Number of RTSS publications in DBLP (data included up to 2012, since 2013 conference will take place in December)	Number of ICSE publications in DBLP (including 2013, since 2013 conference has taken place in May)
Marsha Chechik	-	4 full-length research track papers + 5 shorter papers in other tracks = 9 total
Dirk Beyer	-	4 full-length research track papers + 2 shorter papers in other tracks = 6 total
Steve Goddard	5	-
Abhik Roychoudhury	*7*	*7 full length research track papers*

All papers published in RTSS are full-length research papers. I have co-authored 7 RTSS papers up to 2012, as compared to 5 by Steve Goddard.

For ICSE, the data reported in DBLP counts the main research track papers, and also the short papers such as papers in the ICSE tools track. Counting the papers in main research track of ICSE (where the publication is most prestigious), I have co-authored 7 ICSE papers to-date, as compared to 4 by Marsha Chechik, and 4 by Dirk Beyer.

SABBATICAL LEAVE REPORT: Abhik Roychoudhury
http://www.comp.nus.edu.sg/~abhik

1. **Period of Sabbatical Leave**: 1 August – 31 December 2008.

2. **Summary of Sabbatical Program/Schedule**:

   - Microsoft Research India.
   - Host: Sriram Rajamani, Principal Researcher and Research Manager

3. **Objectives of Sabbatical Program**:

   One of the major objectives of the sabbatical program was to get a more real-world perspective on my research in software validation/debugging. By visiting an industrial lab dealing with extremely large-scale software I wanted to be familiar with the real-life issues in debugging LARGE software systems.

4. **Description of Activities (Academic/Professional/Others):**

   During my visit at MSR, I engaged in intensive discussions and research activities in software debugging. In particular, the discussions focused on debugging of evolving programs. Currently, most software testing/debugging techniques only consider a single program at a time. However, in reality any large scale software system evolves over time, and it is important to take this software evolution into account while performing software testing/debugging. A very common situation that occurs in industrial contexts is: a version of a software system passes all tests and is checked in, and subsequently in a new version, some tests fail. In software engineering terminology, this new-found failure is a *regression,* since the program fails a test which it passed earlier.

   My work at MSR involved developing methods and tools for debugging regressions. Indeed, this is a all-pervasive problem in any software development project --- when a new version of a software is released it may break some tests. Our work takes a fresh look at this problem by employing lightweight program analysis techniques to debug regressions. In the course of our work, we developed a toolkit called DARWIN, named after Charles Darwin who explained the evolution of species. Our DARWIN tool explains software evolution --- given a software regression over two program versions, it highlights portions of the old and new program version which may be responsible for the regression.

   A *technical report* (MSR-TR-2008-91) detailing the research is attached. We

actively engaged with *product groups* in the course of our research work – trying to understand their needs, the in-house tools they currently use, and how our debugging methods can help them in the maintenance of large-scale industrial software systems. MSR has also initiated a *patent* for this work.

After-note: The work was published in ACM SIGSOFT FSE 2009, and the paper was awarded the ACM SIGSOFT Distinguished Paper Award. A US Patent application has been filed by Microsoft.

5. **Whether the research/project can be pursued / developed further on return from sabbatical leave.**

- The research work on testing and debugging of evolving programs, understandably, has many applications. Evolving software appears in many contexts, as discussed in the following.
    - Any software system going through different versions, with new features being added, is an evolving software.
    - In many application domains, such as for security protocols, there is a reference implementation (which serves as the "model"), and several optimized implementations which are actively deployed. So, by using the testing/debugging methods for evolving programs, we can detect/explain potential software attacks. I am currently pursuing this idea in a *research project recently funded by DSTA.*
    - For certain embedded software applications (such as video/audio decoding or encoding), there is a reference implementation C code which is then used for building the actual implementation on a given target architecture. Results from testing / debugging of the reference implementation can then be used for generating suitable test cases of the actual implementation. This is a direction which can be worth pursuing, and given my interests in embedded software, I am thinking along these lines.

6. **Other Benefits**

The research work at MSR has provided a fresh perspective in the field of software and testing in the real-world. Some of the in-house tools in Microsoft (built over many years) made part of the work possible. Apart from impacting my research positively, I also expect this work to influence my teaching and educational work. For example, I am currently finishing up a book on Embedded Software Debugging and Validation, due to be published by Elsevier (Systems on Silicon series) in 2009-10. The research work done at MSR, apart from positively influencing my research, also has had indirect ramifications on my overall take of the software debugging field. This influences my teaching, as well as educational activities such as presentation of software debugging in my upcoming book.

After-note: The book was published by Elsevier in 2009, and has been adopted for teaching world-wide. A Chinese translation of the book has also been published.