Perceptions on the State of the Art in Verification and Validation in Cyber-Physical Systems

Xi Zheng, *Student Member, IEEE*, Christine Julien, *Member, IEEE*, Miryung Kim, *Member, IEEE*, and Sarfraz Khurshid, *Member, IEEE*

***Abstract*—It is widely held that debugging cyber-physical systems (CPS) is challenging; many strongly held beliefs exist regarding how CPS are currently debugged and tested and the suitability of various techniques. For instance, dissenting opinions exist as to whether formal methods (including static analysis, theorem proving, and model checking) are appropriate in CPS verification and validation. Simulation tools and simulation-based testing are also often considered insufficient for CPS. Many "experts" posit that high-level programming languages (e.g., Java or C#) are not applicable to CPS due to their inability to address (significant) resource constraints at a high level of abstraction. To date, empirical studies investigating these questions have not been done. In this paper, we qualitatively and quantitatively analyze why debugging CPS remains challenging and either dispel or confirm these strongly held beliefs along the way. Specifically, we report on a structured online survey of 25 CPS researchers (10 participants classified themselves as CPS developers), semistructured interviews with nine practitioners across four continents, and a qualitative literature review. We report these results and discuss several implications for research and practice related to CPS.**

***Index Terms*—Computational modeling, computer simulation, formal specifications, formal verification, networked control systems, software engineering, software testing.**

I. Introduction

Cyber-physical systems (CPS) feature a tight coupling between physical processes and software components [67] and execute in varying spatial and temporal contexts exhibiting diverse behaviors across runs [107]. CPS are widely used in biomedical and healthcare systems, autonomous vehicles, smart grids, and many industrial applications [67], [90], [107]. Over the years, systems and control engineers have made significant progress in developing system science and engineering methods and tools (e.g., time and frequency domain methods, state space analysis, filtering, prediction, optimization, robust control, and stochastic control) [6]. At the same time, computer science and software engineering researchers have made breakthroughs in software verification and validation. *Validation* assures that a system meets the needs of the customers while *verification* assesses whether a system complies with the specification. Software verification and validation include but are not limited to systematic testing and formal methods. While there are existing well-grounded testing methodologies for other domains of software and formal methods have been used for verification of mission-critical systems in practice, verifying and validating CPS are complicated because of the physical aspects and external environment. For instance, there are insufficient methods for investigating the impact of the environment, or *context*, on a CPS [97]. External conditions, which are often hard to predict, can invalidate estimates (even worst-case ones) of the safety and reliability of a system. Modeling any CPS is further hampered by the complexity of modeling *both* the cyber (e.g., software, network, and computing hardware) and the physical (physical processes and their interactions) [71]. Simplified models that do not anticipate that the physical and logical components fail dependently are easily invalidated.

In a 2007 DARPA Urban Challenge Vehicle, a bug undetected by more than 300 miles of test-driving resulted in a near collision. An analysis of the incident found that, to protect the steering system, the interface to the physical hardware limited the steering rate to low speeds [79]. When the path planner produced a sharp turn at higher speeds, the vehicle physically could not follow. The analysis also concluded that, although simulation-centric tools are indispensable for rapid prototyping, design, and debugging, they are limited in providing correctness guarantees. In some mission-critical industries (e.g., medical devices), correctness is currently satisfied by the documentation for code inspections, static analysis, module-level testing, and integration testing [56]. These tests do not consider the context of the patient [56]. Such a lack of true correctness guarantees could easily cause something like the Therac-25 disaster [70] to reoccur.

We seek to address the dearth of empirical information available about CPS development, specifically in debugging and testing. While limited studies of CPS verification and validation exist [66], [29], [94], there is no study that systematically addresses the entire range of existing approaches. In the past decade, as research on CPS has exploded, many strongly held beliefs have emerged related to developing, debugging, and testing these systems. We conduct a broad literature review, a quantitative survey, and qualitative interviews with CPS experts to uncover the state of the art and practice in CPS verification and validation. Our surveys and interviews start with basic questions, identifying the technical backgrounds of actual CPS experts. We then move into specifics related to

Manuscript received December 27, 2014; revised October 15, 2015; accepted October 19, 2015. This work was supported by the NSF under Grant CNS-1239498.

X. Zheng is with the Center for Advanced Research in Software Engineering, University of Texas at Austin, Austin, TX 78712-0240 USA, and also with the School of Information Technology, Deakin University, Geelong, Vic., 3217, Australia (e-mail: xi.zheng@deakin.edu.au).

C. Julien and S. Khurshid are with the Center for Advanced Research in Software Engineering, University of Texas at Austin, Austin, TX 78712-0240 USA.

M. Kim is with the Department of Computer Science, University of California, Los Angeles, CA 90095-1596 USA.

Digital Object Identifier 10.1109/JSYST.2015.2496293

TABLE I
SUMMARY OF STRONGLY HELD BELIEFS ABOUT CPS DEVELOPMENT

Belief	Sections
I. CPS developers are largely untrained in traditional software engineering methodlogies [34], [90]	IV-A, V
II. CPS developers are generally unfamiliar with traditional software verification and validation tools and methodologies [34], [90]	IV-A, V
III. High-level programming languages (eg., Java) are not applicable to CPS [39], [106]	IV-A, V
IV. Resource constraints (e.g., CPU, memory, and storage) are a major issue in developing and debugging CPS [62], [106], [109]	IV-A, V
V. Existing model checking and other formal techniques are insufficient to meet CPS applications' needs [17], [22], [65], [103]	III, V
VI. Simualtion alone is insufficient in supporting verification and validation of CPS [79]	IV-B, IV-C, V
VII. An ad hoc, trial-and-error approach to development is the state of the art for CPS systems [81], [90]	IV-B, V
VIII. There is a significant gap in language between formal models of computing and communications and models of physics that makes applying them jointly in CPS challenging [2], [67], [96]	IV-B IV-C, V

tools and techniques used on a daily basis. We take a broad view, encompassing simulation, formal methods, model-driven development (MDD), and more *ad hoc* approaches. We also attempt to ascertain what aspects of CPS development remain unaddressed *in practice*, with the aim of eliciting a targeted research agenda for software engineers desiring to support the ever-growing domain of CPS development.

To the best of our knowledge, our study is the first to quantitatively assess the state of the art in this area. We start by identifying strongly held beliefs about CPS debugging (Section II) and then review the relevant available literature (Section III). We follow this with detailed results from an online survey (Section IV) and one-on-one interviews with CPS experts (Section V). We conclude with some future research directions for CPS verification and validation elicited from our investigation (Section VI).

II. Methodology

CPS are increasingly prevalent, and they pervade many other emerging domains, including pervasive computing in general and the Internet of Things. The rise has been so rapid over the past decade that software engineering support for these new domains has not kept pace. We seek to confirm or dispel several widely held beliefs related to developing CPS, with a specific focus on the verification and validation stages (specifically during the testing and other later stages of a software development cycle). Table I documents several of these beliefs, along with relevant references to the literature. The section of this paper in which we address each belief is listed in the right-most column of Table I. Our investigation takes three parts: 1) a broad literature review; 2) a quantitative online survey; and 3) qualitative interviews. This combined study benefits from the strengths of each of its parts; while we feel the study methods are rigorous, some threats to validity still exist. We discuss these in Section VII. For each of our three methods, before discussing the results, we here briefly describe the goals of the approach, our protocol, and how we analyzed the data.

A. Literature Review

It is obviously not possible (or desirable) to perform a *complete* literature review of CPS verification and validation in this paper. Instead we aim to provide a broad look at the variety of techniques and approaches that could be applied to CPS verification and validation. The approaches analyzed in the literature review helped us shape the questions for the online survey and interviews.

Protocol. We conducted this review by exploring related publications in the recent past in the areas/categories of static analysis, theorem proving, model checking, run-time verification, simulation-based testing, synchronous approaches of real-time systems testing, MDD-based tools, and finally social and cultural impact of verification and validation. All of these reviews were focused through a lens capturing CPS and other closely related domains (e.g., hybrid systems and real-time systems). The review reported in this paper is a refinement of a much broader look that included additional domains (e.g., distributed systems in general, reactive systems, and sensor networks) and a deeper look at specific categories of approaches. The artifacts covered in this paper serve as (highly referenced) exemplars of the state of the art in verification and validation for CPS.

Data Analysis. For each category in our review, we chose a few representative approaches (selected based on measures of popularity including citations and discussions of practical applications) and provide a short summary (due to the size limitation) of their pros and cons.

B. Online Surveys

Our survey has two aims. 1) We aim to corroborate findings from the literature review by cross-checking them with those CPS researchers with hands-on experiences. 2) We seek to confirm or dispel the strongly held beliefs listed in Table I.

Protocol. Based on the findings from our literature review, we created a set of multiple choice questions that attempt to resolve the veracity of the strongly held beliefs surrounding CPS development and debugging. We also designed a set of open-ended questions motivated to complement the variety of information collected in the literature review.[1]

We sent the invitation of the online survey to 82 CPS researchers, who publish work related to real-world CPS development and deployment in relevant academic conferences and received 25 responses. We reached experts from a wide range of subfields, including electrical, mechanical, chemical, and biological engineering and from computer science; 37.5% of them have expertise in control systems and AI, 37.5% of them in networking, 16.7% of them in cyber-security, 16.7% of them in civil engineering, mechanical engineering, or other

[1]The surveys were delivered via SurveyMonkey; the full text is available at https://www.surveymonkey.com/s/MP7HP7W.

"traditional" engineering fields, 37.5% in real-time systems, distributed systems, algorithm, verification, testing, and software engineering, in general. When asked about their primary role(s), 70.8% have roles as CPS modeling experts, designers, and architects; 54.2% have roles in validation and verification; and *41.7% classified themselves as CPS developers*. The participants had, on average, 8.35 years of software development experience and 6.69 years of experience in CPS applications.

Data Analysis. We performed statistical analysis on the multiple choice questions. We collated the free-text responses by combining responses that aligned contextually. We use a phenomenological approach [26], which attempts to aggregate meaning from multiple individuals based on their "lived experiences" related to the concept (i.e., phenomenon) under study. Our online survey (and, in fact, our interviews) is exactly targeting the conclusions we can draw based on studies of the experiences of a group of individuals, in this case, experts in CPS development.

C. Interviews

To more deeply examine the implications of several of the responses in the survey and corroborate the findings in the survey relative to the strongly held beliefs listed in Table I, we created open-ended questions around the trends we saw in the survey results, to explore further CPS practitioners opinions related to CPS verification and validation. The full questions list is available online.[2]

Protocol. We conducted these interviews through personal interactions.[3] The audio of the interviews was recorded with the participants' consent. In the case of in-person interviews, the participants often showed the interviewer documents, papers, devices, and other artifacts that were relevant to the interview questions; this often highlighted the real constraints and limitations or showcased the development and deployment environments. The interviewees were CPS experts in charge of real-world CPS systems from around the world (North America, Europe, Asia, and Australia). The real CPS developed and deployed by the interviewees include a large scale bridge health monitoring system in Australia, an assisted living device for elder persons in one of the biggest hospitals in Australia, robots for extra-terrestrial exploration, an autonomous military system, autonomous vehicles in the USA, a ventricular assist device (VAD) in the USA, and a few others. We found the participants through our review of the CPS literature and development tools; the selected interview participants are in charge of the development and deployment of real-world CPS applications development and deployment. We interviewed an expert in *autonomous vehicles*, another in closely related *autonomous robots*, one in *medical CPS*, two in *formal methods*, one in *unmanned aerial vehicles and wireless sensor networks*, one in *assisted living*, one in *wearable devices*, and the last one in *structural health monitoring*.

Data Analysis. We transcribed the interviews and then used the same methodologies as we did for the online survey.

[2]http://goo.gl/5vwvPf

[3]Two interviews were done over Skype; the remainder were in person.

III. Literature Review

In our literature review, we focus on breadth of coverage, providing exemplars in the wide variety of applicable areas, including formal methods, model- and simulation-based testing, run-time verification, and multiple practical tools. We also look briefly at social and cultural factors that have a nontrivial impact on the adoption of these techniques.

A. Formal Methods

Static analysis is used to efficiently compute approximate but *sound* guarantees about the behavior of a program without executing it [33]. Abstract interpretation relates abstract analysis to program execution [25] and can be used to compute invariants [21], [30], [78]; these approaches have been applied in CPS, including in flight control software [12] and outer space rovers [53], [48]. In general, the efficiency and quality of static analysis tools have reached a level where they can be practically useful in locating bugs that are otherwise hard to detect via testing. However, for mission-critical CPS applications (which may contain millions of lines of code that interact in complex ways with a physical world), existing industry static analysis tools either do not scale well (e.g., [53]) or tend to introduce many false positives (e.g., [49], [50], [55]).

Theorem proving has been applied in deductive verification [68], [69], where validity of the verification conditions are determined. Theorem provers have also been used for verifying hybrid systems [1], [82]. Isabelle/HOL [83] has been used to formally verify the kernel piece of seL4 [60], which is the foundation OS for a highly secure military CPS application. This work shows that, with careful design, a (critical component of a) complex CPS can be formally verified by the state of art theorem prover. However, the requirement for human intervention and high costs (the total effort for proof was about 20 person-years, and kernel changes require 1.5–6 person-years to reverify [59]) makes applying theorem proving impossible for general-purpose CPS applications, which may contain millions of lines of code [87] and require much quicker (and less expensive) changes.

The verification world is also rife with highly capable model checkers [28], [63], including those that handle real-time constraints [11], parametric constraints [45], stochastic effects [46], and asynchronous concurrency with complex and/or dynamic data structures (though not sharable between concurrent processes) [38], all of which are common in CPS. Model abstraction and reduction can make analysis more tractable (and thus more applicable to CPS) [24], [41], [104], however, error bounds are usually unquantified, which makes the verification unsafe. While model checking allows verification to be fully automated, in addition to issues such as state-explosion, complexity in property specification, and inevitable loss of representativeness [8], CPS exhibits bugs that crop up only at run-time based on the physical state of the deployment world; such bugs cannot be captured by model checking alone. In hybrid systems, online model-checking has received some attention, investigating, e.g., the potential behavior of a system over some short-term (time-bounded) future. Such approaches have been applied to check medical device applications [17], where the findings have motivated further investigation into adaptations of model checking targeted for CPS-like domains.

It is exceedingly difficult to prove properties of CPS automatically because of the disconnect between formal techniques for the cyber and well-established engineering techniques for the physical [23], [86]. This disconnect is the root of Belief VIII in Table I. In recent works [19], [36], physical and software processes are modeled and composed together either as timed automata or hybrid automata, and the compositional models are verified by model checkers against correctness properties. However, the combinatorial explosion both for the number of reachable discrete states and the reachable sets of continuous variables remains an unresolved research challenge. Furthermore, the large scale of CPS applications pushes scalability requirements well beyond the capabilities of existing tools. Although significant progress has been made in formal verification that has the potential to change this landscape for CPS, without support from other approaches, including runtime verification (which is much less constrained by scalability issues) [10], formal methods alone are not enough to tackle the challenges in CPS verification and validation [22], [65]. These positions from the literature are the root of Belief V in Table I; our survey and interviews will try to further identify uses and challenges associated with real-world CPS developers applying formal techniques.

B. Run-Time Verification

In run-time verification, correctness properties specify all admissible executions using extended regular expressions, trace matches, and others formalisms [10]. Temporal logics, especially variants of LTL [88] are popular in runtime verification. However, basic temporal logics do not capture nonfunctional specifications that are essential in CPS (e.g., timeouts and latency) and lack capabilities to deal with the stochastic nature of many CPS applications [40], [89], [93]. Probabilistic temporal logic such as probabilistic computation tree logic (PCTL) [42] and continuous stochastic logic (CSL) [7] have been introduced two decades ago to specify probabilistic properties, and a subset of CSL [40] and a CSL-compatible language [93] are used to monitor probabilistic properties at runtime. However, these temporal logics lack the capacity to monitor the continuous nature of the physical part of CPS applications. In [100], a monitor is created for stochastic hybrid systems and a monitorability theorem is provided. However, there is little discussion of whether the monitor will impact the system's functional and nonfunctional behaviors. In [61], an efficient runtime assertion checking monitor is proposed for memory monitoring of C programs. This noninvasive monitoring is well suited to mission- critical and time-critical CPS applications. In summary, the state of art in run-time verification can potentially provide a great supplement for formal methods and traditional testing in CPS. However many opportunities remain to make run-time verification more suitable to the idiosyncrasies of CPS and approachable to CPS developers. For instance, aspect-oriented monitoring tools [18] are less intrusive, and their adaptation to CPS run-time verification may prove more approachable for developers.

C. Model-Based Approaches

In this category, we talk about model-based testing, which uses formal models to enable (automatic) testing of CPS applications [105]. We also include simulation-based techniques aimed at the model analysis of CPS applications [13], [37], [72]. Modeling real-time components has been decomposed into behaviors, their interactions, and priorities on them; reasoning can then occur layer by layer [9], [98]. In general, such approaches allow the verification of all system layers from the correctness proof of the lower layers (i.e., gate-level) to the verification procedure for distributed applications; such an approach has been used to verify automotive systems, a key exemplar of CPS [15]. The practicality and costs of development associated with these approaches are still unknown. While there are many computational and network simulators that many software engineers may be familiar with, in the CPS domain, one of the most relevant systems is Simulink, which is widely deployed in the automotive industry and other mission critical domains (e.g., avionic applications) [44]. In [5], A MATLAB toolbox called S-Taliro is created to systematically test a given model by searching for a particular system trajectory that falsifies a given property written in a temporal logic. However, S-Taliro suffers from a memory explosion problem when a system contains larger specification formulas. MATLAB also provides Simulink design verifier (SDV) [54] as an extension toolset to perform exhaustive formal analysis of Simulink models. SDV is able to create test suites satisfying a given model coverage [52], and generate counterexamples for the violation of formal properties (e.g., temporal properties [51]). However, the test inputs are discrete and not suitable for CPS models with time-continuous behaviors, which make SDV an unlikely candidate to capture continuous dynamics of CPS applications [76]. In [31], another MATLAB toolbox called Breach is used for reachability analysis, parameter synthesis, and monitoring of temporal logic formulas. In [32], Breach is able to support time-frequency logic (an extension to signal temporal logic [75]), which is capable of specifying not only temporal logic properties but also frequency-domain properties. However, this toolbox generally requires developers to write supplementary codes to guide the toolset and thus the approach is error-prone. As explored further in our online survey and interviews, the truthfulness of the application's behavior in simulation-based approaches is often in question [66], and in practice extensive simulation can not cope with modeling uncertainty and random disturbances, which are currently only addressed using *ad hoc* methods [6]. As a result when system verification has relied exclusively on simulation, the verification has failed to identify key failure points [79]; in addressing Belief VI from Table I in our survey and interviews, we seek to identify situations when real-world CPS developers rely on simulation and when it falls short.

Model-based approaches are gaining momentum, and it seems inevitable that approaches will emerge that can be applied to general purpose CPS. For now, the high-learning curve associated with creating the models, the costs of developing them, and scalability remain major hurdles to wide adoption.

D. Testing and Debugging Tools

Although sensor networks and CPS are not exactly the same, several tools exist to support testing and debugging deployed wireless sensor networks, which provide insight into directions and challenges for CPS. *Passive distributed assertions* [92] allow programmers to specify assertions that are preprocessed to generate instrumented code that passively transmits relevant messages as the assertions are checked. Dustminer [58] collects system logs to look for sequences of events responsible for faulty interactions among sensors. Clairvoyant [108] uses a debugger on each sensor node to instrument the binary code to enable GDB-like debugging behavior. MDB [102] provides the same style of behavior for *macroprograms* specified at the network level (instead of the node level). Envirolog [73] records all events labeled with programmer-provided annotations, allowing an entire execution trace to be replayed. *Declarative tracepoints* allow the programmer to insert checkpoints for specified conditions that occur at runtime. Sympathy [91] collects and analyzes a set of minimal metrics at a centralized sink node to enable fault localization across the distributed nodes. Finally, KleeNet [95] uses symbolic execution to generate distributed execution paths and cover low-probability corner-case situations. In summary, these tools and algorithms can tackle various similar issues in CPS; an immediate effort to adapt them more specifically to CPS would be one that directly accounts for the physical world with continuous dynamics.

E. Cultural and Social Concerns

To improve the state of art and practice of developing and debugging CPS, it is essential to have a robust, scalable, and integrated toolset that not only provides accessible approaches for verification and validation but approaches that are also more willingly adopted by practitioners. The major cultural and social impediments include *lack of funding* and *lack of priority* [4], which are not related to technical aspects at all. Apart from commonly known false positives, the reasons developers do not use static analysis tools have been documented as *developers' overload* [57], while *mandate from supervisors*, the *(in)ability to find knowledgeable people*, and *code ownership* all play a role in how bugs are fixed [80].

In many of the approaches we reviewed, even when the CPS developers attempted to provide significant rigor to verification and validation tasks, they almost always had to fall back on a "trial and error" approach, which results in a more *ad hoc* approach to verifying the system [81]. In addressing Belief VII from Table I, our survey and interviews seek to uncover how pervasive trial and error is among real-world CPS developers. In correlation with Belief I, we are also interested in whether real-world CPS developers are classically trained software engineers who are aware of more formal methods or whether they are "outsiders" who simply default to *ad hoc* methods.

IV. Online Survey

Our online survey consisted of 32 multiple choice and free answer questions designed to: 1) understand participants' definitions of CPS; 2) determine participants' familiarity with existing techniques for verification and validation, and how they are applied to CPS; and 3) to collect information about the main challenges in verification and validation of CPS, from an "in the trenches" perspective. Table II shows an abbreviated version of the survey. The specific questions were driven by our literature review, which also helped elicit the strongly held beliefs in Table I. The questions were crafted to help confirm or dispel each of these beliefs. Our approach in the survey was *intentional*. We began by generating definitions of both *CPS* and of *verification and validation*. We then built on this foundation to determine, in detail, the experts' various approaches to and perceptions of the wide array of CPS verification and validation techniques.

TABLE II
Summary of Survey Questions

Background
What are your primary application domains of expertise (multiple choice)?
What are your primary roles (multiple choice)?
How many years of CPS development experience do you have?
What programming languages have you used in developing CPS applications? (multiple choice)
Definition
What are the differences between CPS and embedded systems? (free text)
How do you define verification and validation (in general) (free text)?
Perceptions
Please rate agreement or disagreement (strongly agree, agree, neutral, disagree, strongly disagree)
Simulation alone is sufficient for verification and validation of CPS.
Formal methods for verification and validation of CPS is not tractable with respect to resources and time.
The state of the art of verification and validation of CPS involves repeatedly rerunning the system in a "live" deployment, observing its behavior, and tweaking the implementation (both hardware and software) to achieve the stated requirements.
Experience
What percentage of your work is devoted to verification and validation?
How do you think code inspection can help with verification and validation? (free text)
What testing methodologies do you employ during verification and validation of CPS? (multiple choice)
What model checker(s) do you employ during verification and validation of CPS? (multiple choice)
What simulation tools have you used? (multiple choice)
Have you written assertions to aid in verification and validation of CPS?

See https://www.surveymonkey.com/s/MP7HP7W for complete survey.

A. Background and Definitions

Among CPS developers, there are strong opinions about appropriate programming languages. The programming language greatly influences the verification tools and techniques that can be applied; while some techniques apply at the design level and are thus more general purpose, others apply at the language level. The responses to the question "What programming languages are you familiar with?" mirror surveys of programming language adoption in general (with C/C++ and Java taking the top spots, and Python a close third). Only one of our respondents was familiar with nesC, the programming language for TinyOS sensor network platforms. This is interesting given that many CPS experts purportedly believe that high-level

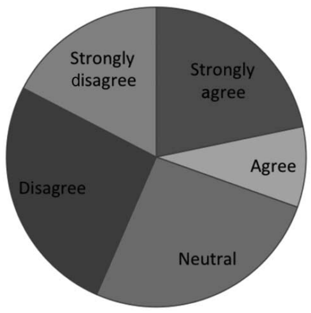

Fig. 1. Programming language like Java is not applicable to systems with hard real-time constraints.

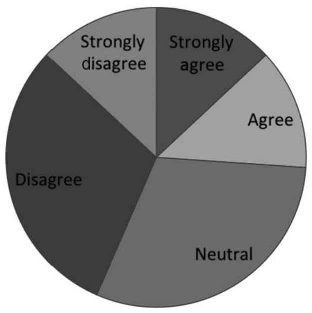

Fig. 2. Use of formal methods for verification and validation of CPS is not tractable with respect to resources and time.

programming languages are not appropriate for cyber-physical style systems [39], [106].

A subsequent question broached this question directly, when we asked participants to rate their agreement with, "A programming language like Java is not applicable to systems with hard real-time constraints." Fig. 1 shows the results; we were surprised by the implication that many of the surveyed CPS experts found Java to be reasonably appropriate for CPS development (50% of self-classified developers, referred as developers, selected disagree/strong disagree, 30% selected neutral). Consider other Java dialects such as RT-Java or Java Embedded, or Java ME, that are designed specifically for developing real-time or embedded applications, this further counters the colloquial claim expressed as Belief III in Table I that high-level languages are not appropriate to CPS development, which has a potential rippling effect on future research directions. In our interviews, we found even stronger evidence for these findings (Section V).

We also asked the participants to express definitions of both CPS and verification and validation in their own words. This is important in setting a foundation for the remainder of the survey responses. When we asked, "In your opinion, what are the main differences between CPS and conventional embedded systems," most respondents' answers identified commonly cited key distinctions; the following responses were typical:

"*embedded systems were mostly focused on software/hardware interacting with low-level sensing and real-time control. CPS includes embedded systems but also networks, security, privacy, cloud computing, and even big data.*"

"*CPSs tend to focus more on the interplay between physical and virtual worlds, and the kind of applications possible with the observation (and modification) of the physical world done through devices embedded in the environment.*"

While the above gets at participants' individual definitions of CPS (which largely converge), we also wanted to understand CPS developers' perspectives on verification and validation. In response to "How do you define verification and validation," over half of the participants gave something quite similar to commonly accepted definitions (i.e., that verification establishes how well a software product matches its specification, while validation establishes how well that software product achieves the actual goal [14]). Many other respondents (30% developers with incorrect answers) failed to correctly express the concepts. Intuitively, these results motivate the creation of easy-to-use tools and better education that enable even CPS developers without a rigorous software engineering background to develop robust systems.

B. Perceptions

One of the primary goals of this survey is to uncover the veracity of the beliefs in Table I. We phrased several of these sometimes controversial points as questions about "perceptions" associated with CPS development. We asked the participants to rate their level of agreement (or disagreement) with the statements using a five-point Likert scale.

It is often stated (and even empirically demonstrated [79]) that simulation does not sufficiently match a system's behavior in the real world. We asked our participants to rate their agreement with "The use of simulation alone is sufficient for supporting verification and validation of CPS." Given the variety of backgrounds among our participants, this question has the potential to tease out a potential dichotomy among CPS developers with different backgrounds. In fact, all but one of the survey respondents selected either "disagree" or "strongly disagree." The one respondent who selected "strongly agree" was also one of the four survey respondents who gave their primary area of expertise as "Civil Engineering/Mechanical Engineering/Other Engineering," where models are more traditionally accepted as complete representations of the system.

Another commonly held belief (Belief V in Table I) is that formal approaches have too high of an overhead to be practically applied in CPS [103]. When we asked the participants to rate "The use of formal methods for verification and validation of CPS is not tractable with respect to resources and time," the diversity of answers was surprising, as was the apparent support for at least limited use of formal methods for CPS. Fig. 2 shows the distribution of responses (40% developers were in favor and 30% selected neutral).

CPS developers will widely claim that the most common approach to debugging CPS requires a significant amount of "trial and error" [81], [90] (Belief VII in Table I). To evaluate this claim, we asked the participants' opinions regarding "The current state of the art of verification and validation of CPS involves repeatedly rerunning the system in a 'live' deployment, observing its behavior, and subsequently tweaking the

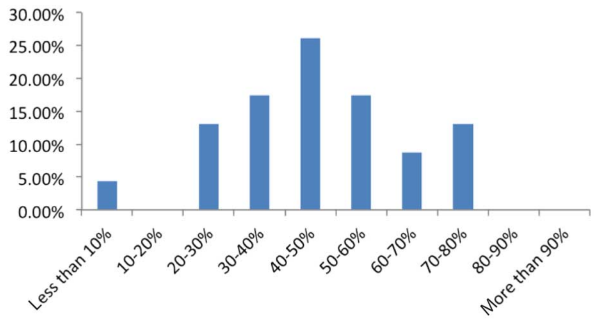

Fig. 3. Project time spent in debugging.

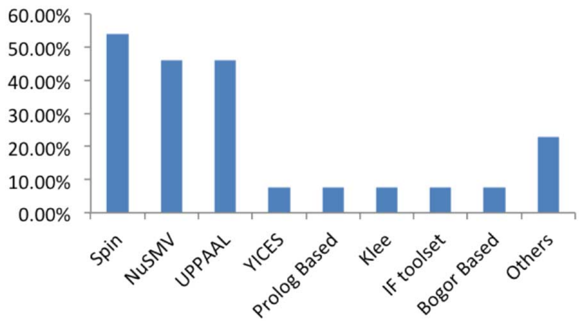

Fig. 4. Model checkers used.

implementation (both hardware and software) to adjust the system's behavior to achieve the stated requirement." 91.3% of the participants expressed either "strongly agree or agree." The current "trial and error" processes are neither rigorous nor repeatable, but the extensive amount of *in situ* debugging that these responses demonstrate motivates better support for approaches to verification and validation that function "in the wild."

Our literature review found that approaches to CPS verification and validation tend to focus either on computational models or on models of physics. Rarely do the two converge. In attempting to address Belief VIII from Table I, the next question in our survey attempted to ascertain whether this is intentional or accidental. We asked the participants to rate their agreement with "A lack of formal connection to models of physics is a key gap in the verification and validation of CPS." We found that 69.6% of the respondents (and 60% of the respondents who also self-identified as CPS developers) selected either "strongly agree" or "agree," while 26.1% (30% of the CPS developers) were "neutral." This is corroborated by a second question, in which we asked the respondents whether they agreed with the statement, "Since CPS has both cyber and physical parts, any approach for verification and validation would need to allow an engineer to, in some way, examine both parts at the same time," to which only 27.3% of the respondents selected "disagree" or "strongly disagree." These two results in conjunction indicate a need for more expressive and integrated models that cross the cyber and physical worlds.

C. Experiences

The third section of our survey queried the participants about their use of verification and validation techniques, most specifically applied to CPS. As Fig. 3 shows, more than 60% of the participants spent between 30%–60% of the system's development time on debugging; more than 20% of the respondents spent *more* time than that. Clearly, debugging CPS is expensive and time consuming.

Only about half of the participants indicated that they employed *code inspection*. Of the respondents who did not use code inspection, the majority found it to be "not relevant." While code inspection is not universally used, it is believed by some developers to provide an important and useful tool to improving code quality and code understanding, which is known to lead to less error-prone implementations [101]. While this motivates better tool support for code inspection of CPS, it is not a clear significant concern of active CPS developers.

We received an evenly balanced response to "Have you used systematic testing to aid in verification and validation?" Participants who responded affirmatively reported improvement of code coverage, systematic review, and identification of corner cases as benefits. Respondents who have not used systematic testing gave standard reasons, including a "lack of time and deep familiarity" and "no easily available tools." Generally, CPS developers are not universally familiar with traditional systematic testing tools. Although systematic testing is well established in more general purpose software engineering domains, there are research challenges in bridging the gap between existing techniques and CPS development.

When we asked "Have you used formal methods (e.g., model checking) to aid in verification and validation?" the majority replied affirmatively. When we followed up with the participants who had employed formal methods about the advantages, they cited inferring useful patterns, complete testing, finding corner cases, and verifying key components. Some participants even reported a sense that model checking was becoming increasingly practical for real systems. Those who did not use model checking said that it is:

"*overly complicated for most purposes; most bugs arise from time dependent interactions with physical systems*";

"*not applicable to my domain; demanding and unreliable.*"

When we asked "What specific model checker(s) do you employ?" participants reported high usage of Spin [47] (53.85%), NuSMV [20] (46.15%), and UPPAAL [63] (46.15%)), as shown in Fig. 4. There was also substantially high use of other (mostly domain-specific) model checkers.

From the free form responses, we noticed that participants gravitate toward general purpose model checkers for very small, very specific pieces of their systems. These model checkers do not enable combined reasoning about the cyber and physical portions of the systems, which is critical to complete and correct verification of CPS.

The responses to "Have you used simulation to aid in verification and validation?" were overwhelmingly positive; only one participant said "no." Participants reported using simulation to understand the system, prototype behavior, refine specifications, explore configurations, and minimize test effort:

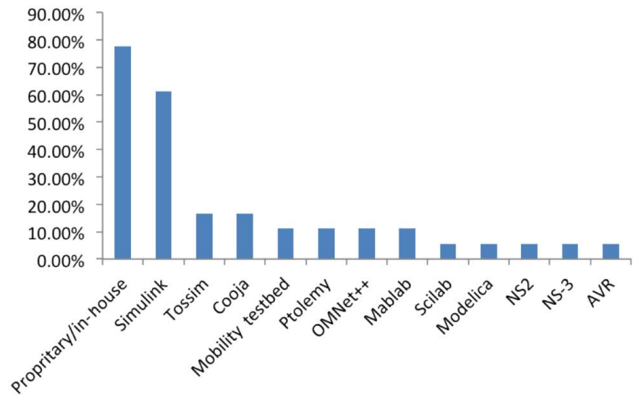

Fig. 5. Simulation tools used.

“*can provide some preliminary confidence of the system*”;

“*can help refine the specification and validate the system*”;

“*allows assumptions made in modeling to be cross-validated against another source of ground truth*”;

“*helps save and focus testing effort.*”

One participant noted, “modeling and simulation only goes so far. No one ever found oil by drilling through a map on a table.” The one participant who did not rely on simulation stated that, “Good enough simulation does not exist.” Participants reported high usage of Simulink (61.1%) and proprietary tools (77.8%), as shown in Fig. 5. The remarkably high use of in-house simulation is concerning because it naturally limits reproducibility and generalizability and implies that developers find that simulation tools in general are not sufficient.

A common approach to debugging at the source level is to augment a program with assertions that provide checkpoints on the program’s state throughout its execution. When we asked the participants about their use of assertions in CPS, the vast majority (more than 70%) had used them. The respondents used assertions primarily for bug detection and writing formal specifications, and, to a slightly lesser extent, to document assumptions. The participants stated that the use of assertions:

“*[provides] formal documentation [and] explicitly states otherwise implicit assumptions, enabling to detect errors sooner and have a better indication of where a problem stems from*”;

“*forces developer to write down (basically as part of the code) the expectation for correct behavior [with] the bonus of being able to ‘execute’ the assertion.*”

The two main reasons cited for not having used assertions in CPS development were concerns about performance and the difficulty in tracing assertions in deployed systems. In general, these results bolster our hypothesis that assertions are a useful means to verify and validate CPS; future research that tailors assertions to particular challenges of CPS (e.g., distribution and physical aspects) may ameliorate some of the concerns.

Finally, we asked our participants’ perceptions of open challenges in verification and validation of CPS. Almost 50% reported issues with physics models, more than a quarter cited scalability, and nearly a quarter reported issues with a lack of systematic verification and validation methods. Fig. 6 reports the complete results. Example statements include:

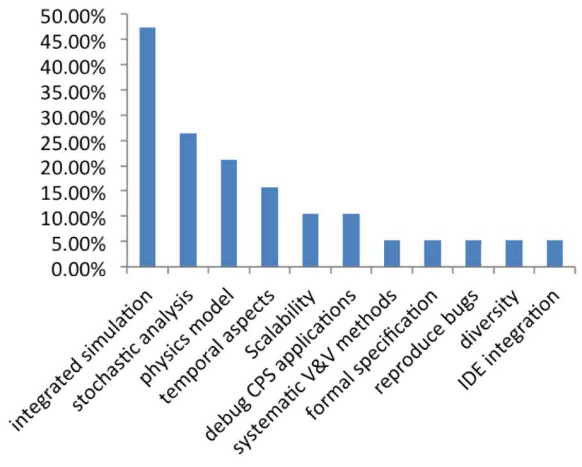

Fig. 6. Main research challenges.

“*CPS models are fragile. We need verification and validation methods that scale with the complexity of the model and are robust to slight variations of the model.*”

“*Time plays a critical role and is misunderstood.*”

“*Impossible to fully understand environment dynamics.*”

The survey results indicate models of software systems, models of physics, and the integration of the two are major bottlenecks in verification and validation of CPS. Scalability of existing techniques and a lack of a capability of directly verifying CPS code from simulation motivate new, tailored approaches that build on and complement the current state of the practice. Before exploring these research challenges, we look at individual interviews with CPS developers.

V. Interviews

The final piece of study is a set of in depth interviews with CPS developers in charge of real-world CPS systems, most of which are mission critical ones (e.g., medical device and bridge structural health monitoring). We were somewhat surprised to find that our survey respondents were not completely against using high-level programming languages like Java to build CPS. The interviews corroborated the survey results and, in fact, highlighted high-level languages that are popular among the CPS developers we interviewed. Specifically, typical responses from our interviewees in response to the discussion question “What are the main programming languages you used in developing CPS applications?” were:

“*For the high level, mostly it is Python. Low level is C and C++. The middle is Java.*”

“*For wireless sensor networks, mainly C, nesC; for aerial drones, mainly C++, Java. We also extended C++, C, and Java with high-level abstractions for specific needs.*”

“*Algorithms developed in MATLAB for modeling and off-line validation; C++ and Java are written on Android phones to reproduce codes in MATLAB*”

"*The sensor [...] software was mainly written in C and C++. The [server software] shown to end users with a web-based system was written in C++ and Ruby on Rails.*"

Quite simply, while "low-level" languages are still popular among CPS developers, high-level languages are also commonly used for CPS development.

We entered our studies with the perception that CPS developers are hampered by severe resource constraints in their deployment environments. Our analysis of the survey results hinted that this might not be the case. In our one-on-one interviews, we discussed the actuality of the resource constraints of the interviewees' target platforms *and* their perceived impact of those constraints on the debugging task. The interview results indicate that CPS developers do not always perceive their target platforms to be resource constrained. Furthermore, the CPS developers we interviewed did not perceive any resource constraints the platforms may have to be a significant impediment to development and debugging:

"*The computation platform is not resource constrained.*"

"*We don not have concerns of resources in general. However, to me, wireless sensor networks are a specific type of CPS, the device nodes are for sure resource constrained. But for other types of CPS applications, it might not be the case.*"

This is an important finding in the sense that techniques for supporting development tasks for CPS (including those for verification and validation) often have a quite significant focus on resource constraints; these efforts may, in fact, be misplaced or at least over-emphasized.

We know from our experience and from the literature that simulation is commonly used in verification and validation in CPS. However, many researchers and practitioners discount the value of simulation. During our interviews, we asked the interviewees about the simulation tools they use and their perception of the pros and cons of using simulation. The following are some samples of the resulting discussions:

"*These simulations are not very truthful. We used an in-house hybrid simulation tool, [but] even with this in-house simulation, we need real testing as the risk is too big for any undetected errors in autonomous vehicles.*"

"*I am not happy with [...] simulation tools as they are not accurate enough; they give you a basic sense of how the system would work, but actually making the simulation work requires [too much effort] to tune parameters.*"

"*We used simulation but what you can test through simulation is only a very small fraction of the problems which can potentially come out when you deploy the system.*"

A common theme was the revelation that the primary simulation tools used were in-house simulators. Further, though from our interviews, we noticed that simulation is increasingly likely to be used primarily only in the earlier stages of design to give a rough view of the system and its behavior.

Concerns about the applicability of model checking to CPS appeared to crop up in our survey; our interviews delved deeper into the use of model checkers by our interview subjects. The responses we received to the question "How do you use model checkers in verifying and validating CPS applications?" indicate that model checkers enjoy only a limited use by in-the-field CPS practitioners, usually employed to check only small pieces of the larger system:

"*We use a very simplistic model for partial ordered sets and use Spin to check it.*"

"*We would like to transform our questions into timed automata and feed the input into UPPAAL. But the model checking suffers from space explosion, and we have to restrict our input to very small set. It is not that useful [...] model checking [does not] fit our needs.*"

The interviewees' comments related to model checking further indicated a desire supplant model checking with more robust run-time verification that is both "online" and "incremental."

We asked, generally, "How do you test CPS applications?" Across the board, the responses validated our view that trial and error is currently the most prevalent approach:

"*We use simulation and trial and error to observe errors.*"

"*Mainly visual observation, look at what robots are doing, take videos and sensor data. Basically, it is trial and error.*"

"*Test software isolated from sensors and controller, then use trial and error to visually observe what is going on.*"

"*Visual observation. We collect traces and print out sensor values. We manually read the traces. It is trial and error.*"

"*We use ground truth and testing to compare results.*"

"*We mainly use automated formal verification and some code review for the kernel part.*"

"*We used volunteers to collect real data and applied them to MATLAB models. The real test is done on real patients. Trial and error. Ground truth is provided by nurses and cameras.*"

The majority of our survey respondents identified a lack of formal connection to models of physics as a key concern. We explored this gap more in the interviews by asking the subjects about the software and physics models they employ. We were surprised to find that CPS developers tend not to deeply consider (formal) models of physical systems during development. They also found available software models inadequate. Some examples of their responses include:

"*The environment is not ideal, we created static physics models to handle noise. The models are still immature and fixed. We need online learning models.*"

"*We used physics models of motors [and a] flow dynamics model. We use these models to determine what forces to counteract using actuation.*"

"*We mainly used distribution models (e.g., partial order models, lattice models) to detect global [correctness] predicates. We abstract away the physics model.*"

Recent emerging work has demonstrated the use of MDD to automatically generate CPS software from heavily validated models [56], [85]. Our interviews attempted to ascertain a practitioner's view on the use of these approaches. MDD, although having a quite lengthy history, is far from mature, especially with respect to CPS. Some examples of our interviewees' responses include:

"*For simple problems, MDD might be possible. But for complex problems, [...] MDD is not very realistic.*"

"*I am not optimistic about this approach. To create models that are very accurate takes too long, which is not useful.*"

"*There is a significant gap between the perceived environment and the modeled environment. [You risk] building a model more complex than the traditional programming task.*"

We also asked our subjects, "How have you used assertions? What improvement you like to see for the use of assertions in CPS?" Our results confirm our intuition that assertions are a reasonable approach to debugging CPS, but that to make them even more appealing, especially to domain experts, an assertion framework should be complemented by CPS-specific features (e.g., temporal and physics aspects).

"*I used assertions to assert the effects of actuation, mainly used for debugging. We actually need to debug assertions, as assertion happens too quickly and it fails to observe the effects of actuation. In CPS, actuation latency is not taken care of by traditional assertions.*"

"*I used assertions to figure out errors. Since I could not step through code since the interaction with the physics, I find assertions is very useful in this regard.*"

Finally, to explore our subjects' opinions on future research directions for CPS development and debugging, we ask open-ended questions, "What are your ideal testing tools for CPS that are not currently available?" The interview subjects described a need for integrated simulation tools, more accessible yet expressive modeling languages, and debugging tools that give programmers greater visibility into the entire system's behavior (both cyber and physical) and better fault localization. The following are direct quotes:

"*high-fidelity simulation with online learning of models.*"

"*accurate run-time models of physics and software models to use for offline development.*"

"*tools that can reproduce bugs.We could throw random errors into the model to check how the system reacts. It is also ideal to have multi-domain models for motors, mechanical systems (for integrated simulation).*"

"*[techniques for] formally specifying behaviors, automating fault localization by specifying a syndrome (e.g., a pattern).*"

"*Theorem proving requires too many human intervention, any more automation can help.*"

"*There are automated code generation from MATLAB models to C++ and Java ready for smart phones. Integration tests have to be done manually; it would be good to have integration test in simulation for heterogeneous models.*"

"*Need a formal specification language for better clarification. Integration test can not be automated between client and server side. [Sensor] node developers and server developers have to manually collaborate for testing.*"

VI. Future Research Directions

From our literature review, survey, and interviews, we have collected a set of potential research directions that have the potential to move CPS debugging into a world where the techniques are more rigorous and repeatable than the *ad hoc* testing that is the current state of the practice.

A. Formal Methods

To analyze continuous aspects of CPS, higher-order-logic automatic theorem provers [43] have been employed. Reducing the enormous amount of user intervention required is a key research challenge. Furthermore, these approaches need to be made more expressive to capture the heterogeneity of CPS. As a future direction, a generic prover supporting other forms of differential equations (i.e., nonhomogeneous) is highly desired. Theorem proving can also be supplemented by static timing analysis to perform program flow analysis, making traditional theorem proving more tenable. Future research in static analysis must deal with the challenges imposed by complex hardware (e.g., multicore platforms with caches) [16]. As for model checking, it would be ideal to have an integrated platform to combine model checkers; for instance, CPS systems could benefit from a combination of a stochastic model checker [23] with KRONOS [28] to explore both stochastic and real-time features of CPS.

B. Simulation

From our online surveys and interviews, a simulation approach that explicitly integrates the cyber and the physical is required. Such *cosimulation* has begun to be explored, for instance, to combine the network simulator ns-2 [99] with Modelica [35] to simulate industrial automation and a power grid [3]. As a step further, CPS developers would benefit from a flexible framework for moving between full simulation (co-simulation) and a full testing environment, allowing aspects of the simulation to be incrementally replaced by physical devices and other characteristics of the real deployment environment. This framework requires an environment in which models and physical devices can "talk" the same language, making the transition from one to the other transparent to the CPS developer and his debugging task.

C. Run-Time Verification

Temporal logics are often used to specify correct system behaviors and used to generate run-time monitors for CPS. However, there are no existing algorithms to generate monitors from metric temporal logic [77], [84]. Combined with the promise that run-time assertions demonstrated in our studies, we expect that a run-time assertion checking framework [110] that captures the essence in MTL (e.g., specifying latency) combined with a high-level modeling language similar to Java modeling language [64] would be more accessible to developers in annotating CPS programs. Such automatically generated monitors would enable CPS validation at run-time in a nonintrusive manner with respect to functional and nonfunctional behaviors of the CPS applications under study.

In general, we should promote solutions that do not interfere with the developer's process (see existing testing methods, for instance). Because the physical world is an essential component of CPS, which means successful approaches will likely function "in the wild."

VII. Threats to Validity

A. Internal Validity

We made some assumptions in some of the findings in Sections IV and V. For instance, from the reported high ratio of in-house simulation, we draw a conclusion that general purpose simulation tools are not sufficient. There might be other confounding variables that result in the high ratio of in-house simulation, for instance participants might have no access to the general purpose simulation tools due to license issues. The online survey's lack of interaction restricted us from ruling out those confounding variables. To mitigate these issues, we used the literature survey and interviews to corroborate our findings. When analyzing interviews and free text answers, we chose to use a phenomenological approach instead of grounded theory [26] because we wanted to attempt to study the *process* of CPS verification and validation and not the *agents* of the process (i.e., the developers themselves) [74]. Grounded theory is also particularly useful if existing theories about the process do not exist [27], which, given our deep literature survey, is clearly not the case here. In analyzing our results, we draw usage conclusions from perceptions about the use (e.g., familiarity with a programming language is indicative of the use of the programming language); again conclusions from the survey were substantively corroborated by the interviews. In our survey, 41% of the CPS researchers also classified themselves as developers. We did not always distinguish results between researchers and these self-classified developers. However, for any questions with significant differences (i.e., the question about the use of formal methods), we did explore these potential two communities for their comparability. Our interviews focused more on practicing CPS developers.

B. Construct Validity

The categories in the literature review and questions in our survey and interviews may neglect important aspects, which may consequently cause us to overlook key issues in verification and validation of CPS. To mitigate this, we carried out an even more in-depth literature study than is reported here; this study covers hundreds of research papers across relevant domains and publication venues. This coverage mitigates the concern that we missed a significant question for our survey or interview. Another construct validity issue lies in the number of participants in the online survey (25) and interviews (9). We did successfully reach a wide cross section of disciplines and cultures, including both researchers (survey) and practitioners (interviews) across the world.

C. External Validity

The participants in the interview are (necessarily) from a limited set of domains. These interviews do not include CPS practitioners from many interesting CPS fields like smart energy grids and smart cities. The conclusions drawn from the interviews may not be applicable to these domains. To mitigate these issue, we did hand pick practitioners across four continents who are directly involved with developing and deploying real (a few large scale) CPS applications from a wide range of subfields.

VIII. Conclusion

We generated an overall picture of the state of the art and state of the practice of verification and validation in CPS though a broad literature survey, an online survey of CPS researchers, and qualitative interviews of CPS practitioners. We focused our investigation around a set of strongly held beliefs associated with the development of CPS. The results for the first two beliefs were mixed: while some CPS developers are deeply familiar with classical software engineering approaches, many are not and even those that are familiar do not apply these techniques generally to CPS. We dispelled the second two beliefs: in fact, high-level programming languages are used by CPS developer experts, and these same experts are not overly hindered by resource constraints. We confirmed that existing formal method techniques and simulation are, as yet, insufficient for supporting the development of entire general-purpose CPS. We also confirmed strongly that the current state of the practice in CPS verification and validation remains an *ad hoc* trial and error process. Finally, we confirmed that there are still significant gaps between the formal models of computing and the formal models of physics that underpin today's CPS systems. This investigation has elicited a set of research directions that have the potential to directly address challenges that real CPS developers cited in the experiences in developing and debugging real-world CPS.

Acknowledgment

The authors would like to thank all participants in the studies.

References

[1] E. Abrahám-Mumm, U. Hannemann, and M. Steffen, "Verification of hybrid systems: Formalization and proof rules in PVS," in *Proc. Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, 2001, pp. 48–57.

[2] R. Akella and B. M. McMillin, "Model-checking BNDC properties in cyber-physical systems," in *Proc. 33rd Annu. IEEE Int. Comput. Softw. Appl. Conf. (COMPSAC)*, 2009, vol. 1, pp. 660–663.

[3] A. T. Al-Hammouri, "A comprehensive co-simulation platform for cyber-physical systems," *Comput. Commun.*, vol. 36, no. 1, pp. 8–19, 2012.

[4] J. Alglave, A. F. Donaldson, D. Kroening, and M. Tautschnig, "Making software verification tools really work," in *Automated Technology for Verification and Analysis*. New York, NY, USA: Springer, 2011, pp. 28–42.

[5] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, *S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems*. New York, NY, USA: Springer, 2011.

[6] R. Baheti and H. Gill, "Cyber-physical systems," in *The Impact of Control Technology*, 2011, pp. 161–166, www.ieeecss.org

[7] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximative symbolic model checking of continuous-time Markov chains," in *Proc. Concurrency Theory (CONCUR'99)*, 1999, pp. 146–161.

[8] T. Ball, V. Levin, and S. K. Rajamani, "A decade of software model checking with SLAM," *Commun. ACM*, vol. 54, no. 7, pp. 68–76, 2011.

[9] A. Basu, M. Bozga, and J. Sifakis, "Modeling heterogeneous real-time components in BIP," in *Proc. 4th IEEE Int. Conf. Softw. Eng. Formal Methods (SEFM)*, 2006, pp. 3–12.

[10] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for LTL and TLTL," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, p. 14, 2011.

[11] M. Ben-Ari, *Principles of the Spin Model Checker*. New York, NY, USA: Springer, 2008.

[12] B. Blanchet *et al.*, "Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software," in *The Essence of Computation*. New York, NY, USA: Springer, 2002, pp. 85–108.

[13] C. D. Bodemann and F. De Rose, "The successful development process with matlab simulink in the framework Of ESA's ATV project," in *Proc. Int. Astronaut. Conf. (IAC)*, 2004, IAC-04-U.3.B.03.

[14] B. Boehm, "Verifying and validating software requirements and design specifications," in *IEEE Softw.*, vol. 1, no. 1, pp. 75–88, Jan. 1984.

[15] J. Botaschanjan *et al.*, "On the correctness of upper layers of automotive systems," *Formal Aspects Comput.*, vol. 20, no. 6, pp. 637–662, 2008.

[16] D. Broman, P. Derler, and J. Eidson, "Temporal issues in cyber-physical systems," *J. Indian Inst. Sci.*, vol. 93, no. 3, pp. 389–402, 2013.

[17] L. Bu *et al.*, "Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior," *ACM SIGBED Rev.*, vol. 8, no. 2, pp. 7–10, 2011.

[18] F. Chen and G. Roşu, "Java-MOP: A monitoring oriented programming environment for Java," in *Proc. Tools Algorithms Construct. Anal. Syst.*, 2005, pp. 546–550.

[19] T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre, "Quantitative verification of implantable cardiac pacemakers," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2012, pp. 263–272.

[20] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A new symbolic model verifier," in *Proc. Comput. Aided Verificat. (CAV)*, 1999, pp. 495–499.

[21] R. Clarisó and J. Cortadella, "The octahedron abstract domain," in *Static Analysis*, Heidelberg, Berlin: Springer, 2004, pp. 312–327.

[22] E. M. Clarke, B. Krogh, A. Platzer, and R. Rajkumar, "Analysis and verification challenges for cyber-physical transportation systems," in *Proc. Nat. Workshop Res. High Confidence Transp.*, 2008.

[23] E. M. Clarke and P. Zuliani, "Statistical model checking for cyber-physical systems," in *Proc. Autom. Technol. Verificat. Anal.*, 2011, pp. 1–12.

[24] R. Colgren, "Efficient model reduction for the control of large-scale systems," in *Proc. Efficient Model. Control Large-Scale Syst.*, 2010, pp. 59–72.

[25] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proc. Principles Programm. Lang. (POPL)*, 1977, pp. 238–252.

[26] J. W. Creswell, *Qualitative Inquiry and Research Design: Choosing Among the Five Traditions*. Newbury Park, CA, USA: SAGE, 2012.

[27] J. W. Creswell and A. L. Garrett, "The "movement" of mixed methods research and the role of educators," *South Afr. J. Edu.*, vol. 3, pp. 321–333, 2008.

[28] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The tool KRONOS," in *Proc. Hybrid Syst. III*, 1996, pp. 208–219.

[29] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling cyber–physical systems," *Proc. IEEE*, vol. 100, no. 1, pp. 13–28, Jan. 2012.

[30] A. Deutsch, "Static verification of dynamic properties," *PolySpace White Paper*, 2004.

[31] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Proc. Comput. Aided Verificat.*, 2010, pp. 167–170.

[32] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, and S. Smolka, "On temporal logic and signal processing," in *Proc. Autom. Technol. Verificat. Anal.*, 2012, pp. 92–106.

[33] V. D'silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1165–1178, Jul. 2008.

[34] J. El-khoury, F. Asplund, M. Biehl, F. Loiret, and M. Törngren, "A roadmap towards integrated CPS development environments," in 1st Open EIT ICT Labs Workshop on Cyber-Physical Systems Engineering, 2013 [Online]. Available: http://bit.ly/1MvTbPQ

[35] H. Elmqvist, S. E. Mattsson, and M. Otter, "Modelica—A language for physical system modeling, visualization and interaction," in *Proc. Comput. Aided Control Syst. Des. (CACSD)*, 1999, pp. 630–639.

[36] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle, "Formal analysis of timing effects on closed-loop properties of control software," in *Proc. Real-Time Syst. Symp. (RTSS)*, 2014, pp. 53–62.

[37] W. Freiseisen, R. Keber, W. Medetz, P. Pau, and D. Stelzmueller, "Using modelica for testing embedded systems," in *Proc. 2nd Int. Model. Conf.*, 2002, pp. 195–201.

[38] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "CADP 2011: A toolbox for the construction and analysis of distributed processes," *Int. J. Softw. Tools Technol. Transfer*, vol. 15, no. 2, pp. 89–107, 2013.

[39] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC language: A holistic approach to networked embedded systems," *ACM SIGPLAN Notices*, vol. 38, pp. 1–11, 2003.

[40] L. Grunske and P. Zhang, "Monitoring probabilistic properties," in *Proc. Eur. Softw. Eng. Conf. (ESEC)/Found. Softw. Eng. (FSE)*, 2009, pp. 183–192.

[41] Z. Han and B. Krogh, "Reachability analysis of hybrid control systems using reduced-order models," in *Proc. Amer. Control Conf.*, 2004, vol. 2, pp. 1183–1189.

[42] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects Comput.*, vol. 6, no. 5, pp. 512–535, 1994.

[43] J. Harrison, *Theorem Proving With the Real Numbers*, in Springer Science & Business Media, 2012.

[44] N. He, P. Rümmer, and D. Kroening, "Test-case generation for embedded Simulink via formal concept analysis," in *Proc. Des. Autom. Conf. (DAC)*, 2011, pp. 224–229.

[45] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: A model checker for hybrid systems," in *Proc. Comput. Aided Verificat.*, 1997, pp. 460–463.

[46] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *Proc. Tools Algorithms Construct. Anal. Syst.*, 2006, pp. 441–444.

[47] G. J. Holzmann, "Basic spin manual," Technical Report, Bell Laboratories, 1994 [Online]. Available: http://spinroot.com/spin/Doc/Manual.pdf

[48] *C Global Surveyor* [Online]. Available: http://ti.arc.nasa.gov/tech/rse/vandv/cgs/, Accessed on: Nov. 2015.

[49] *Coverity* [Online]. Available: http://www.coverity.com, Accessed on: Nov. 2015.

[50] *Klocwork* [Online]. Available: http://www.klocwork.com, Accessed on: Nov. 2015.

[51] *Simulink Design Verifier Temporal Properties—MathWorks* [Online]. Available: http://www.mathworks.com/examples/sldesignverifier/category/temporal-property specification, Accessed on: Nov. 2015.

[52] *Simulink Design Verifier Types of Model Coverage—MathWorks* [Online]. Available: http://www.mathworks.com/help/slvnv/ug/types-of-model coverage.html, Accessed on: Nov. 2015.

[53] *Polyspace Static Analyzers* [Online]. Available: http://www.mathworks.com/products/polyspace/?refresh=true, Accessed on: Nov. 2015.

[54] *Simulink Design Verifier—MathWorks* [Online]. Available: http://www.mathworks.com/products/sldesignverifier/, Accessed on: Nov. 2015.

[55] R. Huuck, A. Fehnker, S. Seefried, and J. Brauer, "Goanna: Syntactic software model checking," in *Proc. Autom. Technol. Verificat. Anal.*, 2008, pp. 216–221.

[56] Z. Jiang, M. Pajic, and R. Mangharam, "Cyber-physical modeling of implantable cardiac medical devices," *Proc. IEEE*, vol. 100, no. 1, pp. 122–137, Jan. 2012.

[57] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, 2013, pp. 672–681.

[58] M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han, "Dustminer: Troubleshooting interactive complexity bugs in sensor networks," in *Proc. SenSys*, 2008, pp. 99–112.

[59] G. Klein *et al.*, "Comprehensive formal verification of an OS microkernel," *ACM Trans. Comput. Syst. (TOCS)*, vol. 32, no. 1, p. 2, 2014.

[60] G. Klein *et al.*, "seL4: Formal verification of an OS kernel," in *Proc. Symp. Oper. Syst. Principles (SOSP)*, 2009, pp. 207–220.

[61] N. Kosmatov, G. Petiot, and J. Signoles, "An optimized memory monitoring for runtime assertion checking of C programs," in *Proc. Runtime Verificat.*, 2013, pp. 167–182.

[62] H. J. La and S. D. Kim, "A service-based approach to designing cyber physical systems," in *Proc. Int. Conf. Comput. Inf. Sci. (ICIS)*, 2010, pp. 895–900.

[63] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *Int. J. STTT*, vol. 1, no. 1, pp. 134–152, 1997.
[64] G. T. Leavens, A. L. Baker, and C. Ruby, "JML: A Java modeling language," in *Proc. Formal Underpinnings Java Workshop (OOPSLA)*, 1998, pp. 404–420.
[65] E. A. Lee, "Cyber-physical systems-are computing foundations adequate," in *Proc. NSF Workshop Cyber-Phys. Syst. (CPS)*, vol. 2, 2006.
[66] E. A. Lee, "Computing foundations and practice for cyber-physical systems: A preliminary report," Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-72, 2007.
[67] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. IEEE Int. Symp. Object Oriented Real-Time Distrib. Comput. (ISORC)*, 2008, pp. 363–369.
[68] K. R. M. Leino, "Efficient weakest preconditions," *Inf. Process. Lett.*, vol. 93, no. 6, pp. 281–288, 2005.
[69] K. R. M. Leino and W. Schulte, "A verifying compiler for a multithreaded object-oriented," *Softw. Syst. Rel. Security*, vol. 9, p. 351, 2007.
[70] N. G. Leveson and C. S. Turner, "An investigation of the Therac-25 accidents," *Computer*, vol. 26, no. 7, pp. 18–41, 1993.
[71] J. Lin, S. Sedigh, and A. Miller, "Towards integrated simulation of cyber-physical systems: A case study on intelligent water distribution," in *Proc. Int. Conf. Dependable Auton. Secure Comput. (DASC)*, 2009, pp. 690–695.
[72] G. Lipovszki and P. Aradi, "Simulating complex systems and processes in LabVIEW," *J. Math. Sci.*, vol. 132, no. 5, pp. 629–636, 2006.
[73] L. Luo, T. He, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic, "Achieving repeatability of asynchronous events in wireless sensor networks with envirolog," Univ. Virginia, Charlottesville, VA, USA, Tech. Rep. ADA447048, 2006.
[74] J. Magnetto Neff, "Grounded theory: A critical research methodology," in *Under Construction: Working at the Intersections of Composition Theory, Research, and Practice*, Logan, Utah State, 1998, pp. 35–124.
[75] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proc. Formal Techn. Modell. Anal. Timed Fault-Tolerant Syst.*, 2004, pp. 152–166.
[76] R. Matinnejad, S. Nejati, and L. Briand, "Automated test suite generation for time-continuous simulink models," Tech. Rep. TR-SnT-2015-7, 2015.
[77] R. Mattolini and P. Nesi, "An interval logic for real-time system specification," *IEEE Trans. Softw. Eng.*, vol. 27, no. 3, pp. 208–227, Mar. 2001.
[78] A. Miné, "The octagon abstract domain," *Higher-Order Symbolic Comput.*, vol. 19, no. 1, pp. 31–100, 2006.
[79] S. Mitra, T. Wongpiromsarn, and R. M. Murray, "Verifying cyber-physical interactions in safety-critical systems," *IEEE Security & Privacy*, vol. 11, no. 4, pp. 28–37, Jul./Aug. 2013.
[80] E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan, "The design of bug fixes," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, 2013, pp. 332–341.
[81] A. Murugesan, S. Rayadurgam, and M. Heimdahl, "Using models to address challenges in specifying requirements for medical cyber-physical systems," in *Proc. Int. Conf. Cyber-Phys. Syst. (ICCPS) Workshops*, 2013, http://www.seas.upenn.edu/~zhihaoj/cps13/Murugesan.pdf
[82] N. Bjoner, Z. Manna, H. Sipma, and T. Uribe, "Deductive verification of real-time systems using STeP," *Theor. Comput. Sci.*, vol. 253, no. 1, pp. 27–60, 2001.
[83] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, vol. 2283, Springer Science & Business Media, 2002.
[84] J. Ouaknine and J. Worrell, "Some recent results in metric temporal logic," in *Proc. Formal Model. Anal. Timed Syst.*, 2008, pp. 1–13.
[85] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam, "Safety-critical medical device development using the UPP2SF model," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4s, p. 127, 2014.
[86] D. L. Parnas, "Really rethinking formal methods," *Computer*, vol. 43, no. 1, pp. 28–34, 2010.
[87] L. Pike, S. Niller, and N. Wegmann, "Runtime verification for ultra-critical systems," in *Proc. Runtime Verificat.*, 2012, pp. 310–324.
[88] A. Pnueli, "The temporal logic of programs," in *Proc. 18th Annu. Symp. Found. Comput. Sci.*, 1977, pp. 46–57.
[89] A. Pnueli, A. Zaks, and L. Zuck, "Monitoring interfaces for faults," *Electron. Notes Theor. Comput. Sci.*, vol. 144, no. 4, pp. 73–89, 2006.
[90] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Proc. Des. Autom. Conf. (DAC)*, 2010, pp. 731–736.
[91] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger," in *Proc. SenSys*, 2005, pp. 255–267.
[92] K. Romer and J. Ma, "PDA: Passive distributed assertions for sensor networks," in *Proc. Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2009, pp. 337–348.
[93] U. Sammapun, I. Lee, and O. Sokolsky, "RT-MaC: Runtime monitoring and checking of quantitative and probabilistic properties," in *Proc. Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, 2005, pp. 147–153.
[94] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-based design for cyber-physical systems," *Eur. J. Control*, vol. 18, no. 3, pp. 217–238, 2012.
[95] R. Sasnauskas, O. Landsiedel, M. H. Alizai, C. Weise, S. Kowalewski, and K. Wehrle, "KleeNet: Discovering insidious interaction bugs in wireless sensor networks before deployment," in *Proc. Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2010, pp. 186–196.
[96] J. Shi, J. Wan, H. Yan, and H. Suo, "A survey of cyber-physical systems," in *Proc. Wireless Commun. Signal Process. (WCSP)*, 2011, pp. 1–6.
[97] S. Sierla, B. M. O'Halloran, T. Karhela, N. Papakonstantinou, and I. Y. Tumer, "Common cause failure analysis of cyber-physical systems situated in constructed environments," *Res. Eng. Des.*, vol. 24, no. 4, pp. 375–394, 2013.
[98] J. Sifakis, "A framework for component-based construction," in *Proc. Int. Conf. Softw. Eng. Formal Methods (SEFM)*, 2005, pp. 293–299.
[99] N. Simulator, "ns-2," 1989.
[100] A. P. Sistla, M. Žefran, and Y. Feng, "Runtime monitoring of stochastic cyber-physical systems with hybrid state," in *Proc. Runtime Verificat.*, 2012, pp. 276–293.
[101] H. Siy and L. Votta, "Does the modern code inspection have value?," in *Proc. Int. Conf. Softw. Maintenance (ICSM)*, 2001, p. 281.
[102] T. Sookoor, T. Hnat, P. Hooimeijer, W. Weimer, and K. Whitehouse, "Macrodebugging: Global views of distributed program execution," in *Proc. SenSys*, 2009, pp. 141–154.
[103] R. A. Thacker, K. R. Jones, C. J. Myers, and H. Zheng, "Automatic abstraction for verification of cyber-physical systems," in *Proc. Int. Conf. Cyper Phys. Syst. (ICCPS)*, 2010, pp. 12–21.
[104] A. Tiwari and G. Khanna, "Series of abstractions for hybrid automata," in *Proc. Hybrid Syst. Comput. Control*, 2002, pp. 465–478.
[105] J. Tretmans and E. Brinksma, "Torx: Automated model-based testing," 2003.
[106] K. Wan, D. Hughes, K. L. Man, and T. Krilavicius, "Composition challenges and approaches for cyber physical systems," in *Proc. Int. Conf. Netw. Embedded Syst. Enterp. Appl. (NESEA)*, 2010, pp. 1–7.
[107] K. Wan, K. Man, and D. Hughes, "Specification, analyzing challenges and approaches for cyber-physical systems (CPS)," *Eng. Lett.*, vol. 18, no. 3, pp. 308, 2010.
[108] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse, "Clairvoyant: A comprehensive source-level debugger for wireless sensor networks," in *Proc. SenSys*, 2007, pp. 189–203.
[109] Y. Zhang, I.-L. Yen, F. B. Bastani, A. T. Tai, and S. Chau, "Optimal adaptive system health monitoring and diagnosis for resource constrained cyber-physical systems," in *Proc. Int. Symp. Softw. Reliab. Eng. (ISSRE)*, 2009, pp. 51–60.
[110] X. Zheng, "Physically informed assertions for cyber physical systems development and debugging," in *Proc. PERCOM Workshops*, 2014, pp. 181–183.

Xi Zheng (S'12) received the Bachelor's degree in computer information systems from FuDan University, Shanghai, China, in 2001, the Master's degree in information science from the University of New South Wales, Sydney, N.S.W., Australia, in 2006, and the Ph.D. degree in software engineering from the University of Texas at Austin, Austin, TX, USA, in 2015.

He is a Research Fellow in Cyber Physical Systems with Deakin University, Geelong, Vic., Australia. Before pursuing the Ph.D. degree in USA, he has been working as a Senior Principal Consultant and Solution Architect in Sydney for over 8 year. His research interests include the design and implementation of middlewares for cyber physical systems (CPS) and Internet of Things in general.

Christine Julien (M'03) received the B.S. degree in computer science, the M.S. degree in biology, D.Sc. degree in computer science from Washington University in St. Louis, St. Louis, MO, USA, in 2000, 2003, and 2004, respectively.

She is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA. Her research has been supported by the National Science Foundation (NSF), the Air Force Office of Scientific Research (AFOSR), the Department of Defense and Freescale Semiconductors. The work has been recognized by an NSF CAREER Award and an AFOSR Young Investigator Award, and the results have appeared in many peer-reviewed journal and conference papers. Her research interests include the intersection of software engineering and dynamic, unpredictable networked environments.

Miryung Kim (M'04) received the B.S. degree in computer science from Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2001, and the M.S. and Ph.D. degrees in computer science and engineering from the University of Washington, Seattle, WA, USA, in 2003 and 2008, respectively.

She is an Associate Professor with the Department of Computer Science, University of California, Los Angeles, CA, USA. Between January 2009 and August 2014, she was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA.

Dr. Kim was the recipient of an NSF CAREER Award in 2011, a Microsoft Software Engineering Innovation Foundation Award in 2011, an IBM Jazz Innovation Award in 2009, a Google Faculty Research Award in 2014, and an Okawa Foundation Research Grant Award in 2015.

Sarfraz Khurshid (M'99) received the B.Sc. degree in mathematics and computer science from Imperial College London, London, U.K., in 1997; read Part III of the Mathematical Tripos at Trinity College Cambridge, Cambridge, U.K., and the Ph.D. degree in computer science from Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2004.

He is an Associate Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA, where he leads the Software Verification and Testing Research Group. He is an Avid Player and Keen Follower of squash (the racket sport, not the vegetable). His research interests include software testing, model checking, specification languages, code conformance, data structure repair, and parallel and incremental techniques for software analysis.

Dr. Khurshid was the recipient of the ACM SIGSOFT Impact Paper Award for 2012, two ACM SIGSOFT Distinguished Paper Awards (ISSTA 2002 and ICSE 2010), a Best Research Paper Award (ASWEC 2009), and an NSF CAREER Award (2009).