Towards an Engineering Discipline for Resilient Cyber-Physical Systems

Ricardo D. Caldas
ricardo.caldas@chalmers.se
Chalmers University of Technology, Gothenburg, Sweden

ABSTRACT
Resilient cyber-physical systems comprise computing systems able to continuously interact with the physical environment in which they operate, despite runtime errors. The term resilience refers to the ability to cope with unexpected inputs while delivering correct service. Examples of resilient computing systems are Google’s PageRank and the Bubblesort algorithm. Engineering for resilient cyber-physical systems requires a paradigm shift, prioritizing adaptability to dynamic environments. Software as a tool for self-management is a key instrument for dealing with uncertainty and embedding resilience in these systems. Yet, software engineers encounter the ongoing challenge of ensuring resilience despite environmental dynamic change. My thesis aims to pioneer an engineering discipline for resilient cyber-physical systems. Over four years, we conducted studies, built methods and tools, delivered software packages, and a website offering guidance to practitioners. This paper provides a condensed overview of the problems tackled, our methodology, key contributions, and results highlighting. Seeking feedback from the community, this paper serves both as preparation for the thesis defense and as insight into future research prospects.

KEYWORDS
Resilience, Cyber-Physical Systems, Software, Doctoral Thesis

1 INTRODUCTION
Cyber-Physical Systems (CPSs) unite the digital and physical worlds, increasingly supporting individuals and groups in their social and professional endeavors. In contrast to traditional embedded systems, CPSs are often designed as networks of interactive and dynamic elements [24], including healthcare systems, mobility systems, process control systems, and robotics. Such applications directly reflect strategic economic and social development areas, i.e., transport, energy, well-being industry, and infrastructure. The European roadmap and strategy for cyber-physical systems (CyPhERS [52]) lists five challenges to the development of CPSs: interoperability, autonomy, privacy, resilience, and uncertainty. Although all these challenges are important and interconnected, we focus on discussing resilience.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

FSE'24, July 15–19, 2024, Porto de Galinhas, Brazil
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nmnmmnn.mnnmnn

Resilience is “the persistence of service delivery that can justifiably be trusted, when facing changes.” [39]. Archetypal examples of resilient computing systems are Google’s PageRank [16, 26] and the Bubblesort algorithm when compared to more efficient MergeSort or QuickSort [2]. Engineering resilient CPSs involves rethinking their design focused on dynamic environments. CPSs are subject to interactions with (human) users and operators [28], changing needs [7], and heterogeneous components joining and leaving the system [29]. Therefore, the software controlling the CPS operation must automatically manage dynamic interactions and uncertain environments [3, 58], with guarantees of compliance with the system requirements [20, 57]. Software engineers, however, face the challenge of guaranteeing that complex systems (e.g., CPSs) are dependable to dynamic changes in the system and the environment [47].

Our study argues that resilience must be embedded in CPS by design and through testing. Therefore, we aim to develop a software discipline focusing on software-centered resilience design for CPS, utilizing software-aided verification and validating CPS with realistic scenarios. To this end, we developed open-access artifacts that solve fundamental barriers to engineering resilient CPS: scientific studies [11–13, 17, 41, 45, 50, 51, 55], scientific artifacts [6, 27], software packages [5, 44, 48, 49], and a website with guidelines [10]. Our research aims to significantly contribute to Software Engineering by focusing on resilient CPS as a future software-centered discipline [34, 37, 42, 54].

2 METHODOLOGY
Our research goal is to step towards addressing the challenge of ensuring CPS resilience amidst dynamic system and environmental changes. To this end, we developed and delivered open-access artifacts that solve fundamental barriers to engineering resilient CPS. We tackled challenges stemming from designing, implementing, and testing resilient CPS. The artifacts resulted from an incremental and cyclic process named design science [30], the process consists of attaining awareness about a problem, devising a solution, and validation. To attain awareness about the problem we used expansive inquiry combined with literature reviews [36], and mapping studies [43]. Then, we developed solutions by creatively composing existing scientific artifacts. Finally, to validate the creative process employed in the solution step we mainly used experimentation [59] and statistical analyses [4, 21]. Lessons learned and reflections from a validation effort were used to sharpen recurring cycles of awareness attainment and solution refinement.

3 CONTRIBUTIONS AND RELATED WORK
We provided the artifacts supporting key activities in establishing a software discipline for resilient CPS engineering depicted in Fig. 1.
This section expands on how our contributions answered research questions to mitigate barriers that hamper research towards the engineering discipline.

**RQ1: How to design efficient self-adaptation for CPSs?** Designing for fail-safe operation requires enabling self-adaptation to adequate the system behavior to the specification goals continuously [3, 58]. To this end, software engineers must design the system to monitor, reason, and act to change itself [35]. Yet, lack of engineering knowledge, large input spaces, and component coordination threaten such a design. Consequently, the design of efficient self-adaptation requires design decisions at the architecture level and component level. At the architecture level, we contributed to enabling and facilitating dynamic analysis processes, i.e., runtime verification and field-based testing. To this end, we prepared guidelines [10, 12] that discuss constraint identification, design patterns, and instrumentation techniques. To manage large adaptation space posed by the uncertain environment, we proposed a hybrid architecture for search-based control-theoretic controller design [13]. Differently from other works that use learning, online or offline, and other statistical methods to reduce the adaptation space [25, 32, 46], improves the efficiency of synthesizing adaptation strategies by optimizing the search process exploring the adaptation space. Our approach proved efficiency in three adaptation scenarios on a healthcare system, the BSN [27, 48]. At the component level design [1, 33] we proposed Mission Control [51] leveraging ensembles to coordinate heterogeneous CPSs components to solve complex missions through efficient coalition formation. Our work evaluated Mission Control using the RoboMAX scenarios [5, 6] to show efficiency in dealing with uncertain scenarios.

**RQ2: How to formally ensure whether the adaptive CPS complies with the specifications?** Formally ensuring adaptive behavior requires code-level interventions, formal verification, and defect identification. Model-based code-level interventions [9, 18, 38] enforce environment assumptions in code but demand an additional modeling step. We introduced four design patterns for embedded domain-specific languages [56] to offer code-level assurances to CPSs [49, 50]. We also contributed a mapping between software properties in LTL (Linear Temporal Logic) and control quality attributes [11] as an enabler to checking control-theoretic properties in code. Differently from Cámara et al. [14], our mapping technique relies on property specification patterns. We recommend how to generate monitors, prepare the execution environment, instrument the system, execute the system, and post-mortem analysis[10, 12]. Regardless, we proposed a search-based algorithm that takes an execution trace and a violated property to find a diagnostic for what has violated the trace [41]. Unlike works that extrapolate information from trace-slicing [22, 23] or showing common behaviors [19, 40], our diagnostics algorithm explains violations through the mutations applied to the property. In another work, we proposed an immune-inspired negative selection algorithm for detecting candidate features in a violated property [17].

**RQ3: How to validate large and complex CPS operating in realistic scenarios?** CPSs are often large and complex systems and must handle complex human interactions and making test campaigns hard to implement, even harder to automate [8]. For this reason, testing CPS in realistic scenarios is costly and slow. The activity requires rigorous planning yet there are no best practices to support field-based testing campaigns [31]. Consequently, we derived recommendations on how to specify the (un-)desired behavior, generate test cases and prepare oracles, instrument the CPS, collect data, and generate reports[10, 12]. Also, we developed human-vehicle models for scenario-basted testing [44, 45]. In complement to current vehicle simulations, e.g., macroscopic [53] or microscopic models [15], our human-vehicle models compose maneuver models consisting of accurate algebraic implementations. Additionally, we contributed PASTA, a testing technique that uses markov models to simulate patients by considering sensor data trends captured by a networked healthcare system [55]. Our method found bugs in the BSN [27, 48] more efficiently than random generation.

---

**4 CONCLUSION**

Our research aims to significantly contribute to Software Engineering by focusing on CPS resilience as a future software-centered discipline. To this end, we followed design science to develop methods and tools to design for efficient self-adaptation, formally ensure that the CPS complies with the specifications, and validate the CPS in realistic scenarios. Ultimately, our study benefits researchers and practitioners in Software Engineering.