Thesis for The Degree of Doctor of Philosophy

Behavior Trees for Robotics Software Engineering

Razan Ghzouli

Division of Interaction Design and Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2026

Behavior Trees for Robotics Software Engineering

Razan Ghzouli

ISBN 978-91-8103-484-4
Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 5941.
ISSN 0346-718X

Department of Computer Science & Engineering
Division of Interaction Design and Software Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2026.

"It Takes a Village to Get a PhD...."
D. Kadioglu & C. Valli

Abstract

Context. Robotic systems are used more widely in homes and industry, but their often ad hoc development methods reduce the reliability needed for ongoing growth. Recently, more robotics experts have adopted software engineering best practices to improve system reliability. Building on previous work, we aim to further support the development of reliable robotics missions.
Objective. The objective of this thesis is to build systematic knowledge from empirical insights into the current state of practice in specifying robotics missions, and to develop solutions that support improved software engineering practices for the development of robotic missions based on our accumulated knowledge.
Method. This thesis employs a mixed-methods approach, combining empirical and constructive research methodologies. The empirical phase involved knowledge-seeking studies that employed comprehensive analyses of behavior modeling languages and their domain-specific languages, mined open-source projects, and conducted action research and survey studies with practitioners. The constructive research phase involved solution-seeking studies employing the accumulated knowledge to design, implement, and evaluate two solutions in collaboration with practitioners to problems identified in the previous phase. Our research was grounded in the context of robotics software engineering and, when possible, conducted in collaboration with both academic and industrial practitioners.
Result. The empirical contributions provided a comprehensive understanding of behavior trees as a behavior modeling language for robotics missions. This included comparisons with UML behavior modeling languages and state machines, as realized in robotics, regarding modeling concepts and language design, their adoption in open-source robotic projects, and practitioner practices and experiences. Based on these insights, two solutions were developed and evaluated with practitioners. The first introduces a meta-model extension and domain-specific language for behavior trees, integrated into a popular library, to specify quality concerns. The second proposes an approach for supporting industrial risk assessment by integrating behavior trees, enabling early identification of risk concerns and ensuring that their outcomes are adequately considered in the software implementation of robotics missions.
Conclusion. This thesis advances empirical understanding of behavior modeling languages for specifying robotics missions, focusing on behavior trees, an emerging model for behavior coordination and execution in robotics. The built knowledge about behavior trees characterizes their support, usage, and practitioner perspectives. The developed solutions show behavior trees' potential beyond coordination and execution, using them to capture quality concerns and support risk assessment, thereby opening new directions for their role within the broader robotic software engineering lifecycle.

Keywords

Behavior trees, Robotics, Software engineering, Empirical research, Model-based engineering

Acknowledgment

I want to acknowledge everyone who has supported me throughout my PhD journey. My thesis is not only the result of years of my dedication and tremendous effort, but also the village support behind me. This work is supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Without the WASP funding and the great opportunities, none of this work would be possible. I want to extend my appreciation to my different supervisors through out the PhD years, Rebekka Wohlrab, Daniel Strüber, Jennifer Horkoff, Patrizio Pelliccione, and Thorsten Berger. Every one of you contributed to my journey and I learned something from you; thank you. To all my collaborators on my papers, especially Andrzej Wąsowski, Einar B. Johnsen, and Ricardo D. Caldas, I learned a lot from you; thank you. To the current and previous members of the PhD school, especially Agneta Nilsson, Wolfgang Ahrendt and Clara Oders; thank you for supporting me when I needed the most. To my colleagues and those who became friends along the way; thank you. To my family, Mom, Dad, my sister, little nephew and brother, our calls made PhD life easier; thank you. Finally, to my partner in life and amazing supporter, Elias, without you I wouldn't have made it; thank you.

List of Publications

Appended publications

This thesis is based on the following publications:

[A] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, A. Wasowski "Behavior Trees in Action: A Study of Robotics Applications"
Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering (SLE), 2020.

[B] R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, S. Dragule "Behavior Trees and State Machines in Robotics Applications"
Published in IEEE Transactions on Software Engineering journal (TSE), 2023.

[C] R. Ghzouli; J. Horkoff; D. Strüber; R. Wohlrab "Behavior Trees for Robotic Systems: An Empirical Study on Practices and Experiences"
Under review at Empirical Software Engineering journal (EMSE), 2026.

[D] R. Ghzouli; R. Wohlrab; J. Horkoff "Extending Behavior Trees for Robotic Missions with Quality Requirements"
Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), 2025.

[E] R. Ghzouli; A. Hanna; E. Erös; R. Wohlrab "Using Behavior Trees in Risk Assessment"
Proceedings of the IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA), 2025.

Other publications

The following publications were published during my PhD studies. However, they are not appended to this thesis, due to contents not related to the thesis.

[a] A. Bergel, R. Ghzouli, T. Berger, M. RV. Chaudron "FeatureVista: interactive feature visualization"
Proceedings of the 25th ACM International Systems and Software Product Line Conference (SPLC), 2021.

[b] R. Caldas, R. Ghzouli, A. V. Papadopoulos, P. Pelliccione, D. Weyns, T. Berger "Towards Mapping Control Theory and Software Engineering Properties using Specification Patterns"
2nd International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). IEEE, 2021.

Research Contribution

In this section, I describe my contributions to the appended papers. In paper A, I led the conceptualization and the formulation of the methodology. I implemented the data collection of behavior tree literature, domain-specific languages, and projects. Additionally, I contributed to the development of the comparison with UML models. I conducted both quantitative and qualitative analyses of the projects and reported my observations. Finally, I led the writing of the paper and coordinated collaboration with my co-authors.

In paper B, I led the conceptualization and the formulation of the methodology. I led and implemented the entire data collection of state machine literature and DSLs. I managed the data collection of state machine projects. I led and reported the comparison between behavior trees and state machines. I conducted the quantitative and qualitative analysis of projects and reported my observations. Finally, I led the paper writing and coordinated the collaboration with my co-authors.

In paper C, I led the conceptualization and the formulation of the methodology. I regularly visited the company and collected data. I analyzed the data obtained from the action research and subsequently designed the survey based on these results. I analyzed and presented the data in the paper. Finally, I led the writing of the paper and coordinated collaboration with my co-authors.

In paper D, I led the conceptualization and the formulation of the methodology. I led the design of the meta-model extension and developed the proposed solution in Eclipse. I designed and conducted the preliminary evaluation and analyzed the resulting data. Finally, I led the writing of the paper and coordinated collaboration with my co-authors.

In paper E, I led the conceptualization and the formulation of the methodology. I regularly visited the company to develop the approach, collect data, and conduct the internal evaluation. I designed and facilitated the external evaluation. I conducted the data analysis and reported the findings. Finally, I led the writing of the paper and coordinated collaboration with my co-authors.

Contents

Chapter 1

Introduction

Robotics systems are increasingly shaping various sectors of industry and society. They are being deployed to perform a wide range of missions, from pick-and-place in factories to disinfecting hospitals. Even as early as 2017, a survey of over one thousand CEOs worldwide indicated that 52% of surveyed CEOs were already exploring opportunities and benefits associated with integrating machines to collaborate with humans [1]. The inherent complexity of robotic systems arises from the need to integrate multiple hardware and software components to achieve desired functionality. The widespread deployment of robotic systems across various sectors, particularly in environments shared with humans, has heightened environmental uncertainty and increased the challenges associated with developing robotic missions. As robotic missions become more complex and further integrated into human contexts, additional research is required to ensure the safety and reliability of these systems.

Unreliable robotic systems can result in catastrophic incidents, causing injury to humans and damage to property. For example, delivery robots operated by a private company have, on separate occasions, collided with bus shelters in Chicago. These incidents have prompted debate regarding the readiness of such systems for widespread deployment in public spaces, leading to increased public concern and closer regulatory monitoring [2]. Delays in production and a rise in incidents are among the consequences of unreliable robotic systems, potentially resulting in the suspension or withdrawal of robotic deployments. In addition to harming individuals and property, unreliable robotic systems may also adversely affect future economic growth. According to PwC [3, 4], the adoption of smart technologies, including robotic systems, is projected to have a significant impact on economic growth, with estimates suggesting that these technologies could contribute up to 14% to global GDP by 2030, equivalent to approximately $15 trillion at current values. Therefore, enhancing the reliability of various components within robotic systems is essential for realizing both economic and social benefits.

Throughout this thesis, our goal is to support the development of reliable robotic missions by applying software engineering practices. To accomplish this, we pursue two objectives. First, we provide empirical insights into the current state of practice in specifying robotics missions, thereby establishing practical knowledge that informs improved solutions and methodologies. Sec-

ond, we develop solutions that address identified practical gaps, leveraging our accumulated knowledge to support the creation of reliable robotics missions. In our studies, we leverage model-based and model-driven engineering techniques, separation of concerns and roles, explicit requirements specification, and traceability between risk analysis and implementation artifacts as software engineering practices.

The thesis is organized as follow: Section 1.2 familiarizes the reader with key concepts used later in the thesis and related work where possible in, Section 1.1 establishes the motivation behind our work, Section 1.3 presents the RQs guiding the thesis, Section 1.4 discusses the followed methodology to answer the RQs, Section 1.5 discusses the answers of the RQs and their implications, and finally Section 1.7 provides a concluding overview about the thesis and future directions.

1.1 Research Motivation

Robotic systems are increasingly deployed to perform domestic and industrial tasks. With this wide deployment, it is becoming essential more than ever to have fast and reliable development of robotic systems. Developing robotic systems is inherently multidisciplinary and complex requiring collaboration among experts from different domains and combining multiple hardware and software components. Despite the rapid deployment of robotic systems, the maturity of robotic software development has lagged behind. Practices differ significantly across teams and software researchers often described robotic systems as ad hoc [5, 6, 7]. According to the robotic 2020 multi-annual report ICT-2017, a shift towards adopting software practices is needed to manage the complexity of robotic systems [6].

In response, the last two decades have seen a shift in the robotic community to adapt some of the best practices from different domains to support fast and reliable robotic systems. The Robot Operating System (ROS) is the most prominent example. ROS was created as a collective effort by industrial and research practitioners to create an open-source robot operating system that provides support for reusing code and building scalable robotic software systems [8]. As the research around ROS was growing, the community around it did as well. The collective effort of the community developed supporting tools with the aim to improve robotic software systems [9], growing ROS into one of the de facto platforms for robot software. Later, ROS was re-engineered as ROS 2 to meet the design, architecture, and real-world deployment requirements not anticipated by the original version [10].

Examining other solutions that have been developed by industrial and research practitioners in robotic software systems, we observe an interesting trajectory illustrating how robotics software matures. Solutions emerge from practice, and community-driven adoption and research for understanding their strengths and limitations feeds the next generation. Motivated by similar trajectory, the aim of this thesis to support the maturity of developing reliable robotic missions.

Behavior modeling languages are at similar trajectory as ROS. Researchers have noted that the majority of solution-seeking research in behavior modeling

languages are only applied in the lab or applied to simple examples [11, 12]. As such, additional work needs to be conducted to make them usable in practice and everyday life [12]. Within this landscape a specific behavior modeling language called behavior trees (BTs) has emerged. BTs have been developed and widely embraced by the robotic community for specifying robotic behavior, thus, they are promising for further investigation. [13]. One of the Projects that have been funded by the European Union's Horizon 2020 Research and Innovation Program under the RobMoSys project umbrella focused on creating tools for using BTs [14].

Despite this popularity, BTs remain comparatively under-studied from a software engineering lens. This thesis therefore takes BTs as a focal behavior modeling language for supporting reliable robotic missions to create an empirical understanding and build solutions. We want the developed knowledge and understanding in this thesis to inform not only BTs themselves, but the design and evaluation of future behavior-modeling languages in robotics software engineering, much as the study of ROS informed its successors.

1.2 Background & Related Work

In the following, we introduce key concepts used in this thesis and relate our work to the existing body of research.

1.2.1 Model-Driven and Model-Based Engineering

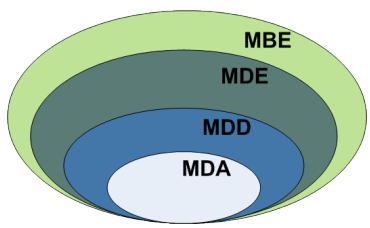

Figure 1.1: The relation between the different modeling terms in software engineering. Figure origin from [15].

This thesis uses the terms model-based (MB) and model-driven (MD) engineering. Model-driven engineering is often conflated with model-based engineering. To clarify these terms, this thesis adopts the definitions and illustrations provided by Brambilla et al. [15]. Figure 1.1 presents an overview of the relationships among various modeling terms in software engineering, as described in [15]. Model-based engineering (MBE) incorporates models within the development process, though models may not necessarily be the primary artifacts. In contrast, model-driven engineering (MDE) treats models as the central artifacts of development. Additional approaches include model-driven development (MDD), in which code is generated semi-automatically from models, and model-driven architecture (MDA), the standardized form of MDD promoted by the Object Management Group (OMG). MDA enables the

definition of models at multiple levels of abstraction, and promotes separating platform-independent models (PIM) from platform-specific models (PSM), as illustrated in Fig. 1.2. It is out of our scope to discuss the terms in detail. Our focus throughout the thesis remains on leveraging models to improve robotic systems, irrespective of whether models serve as the primary artifacts.

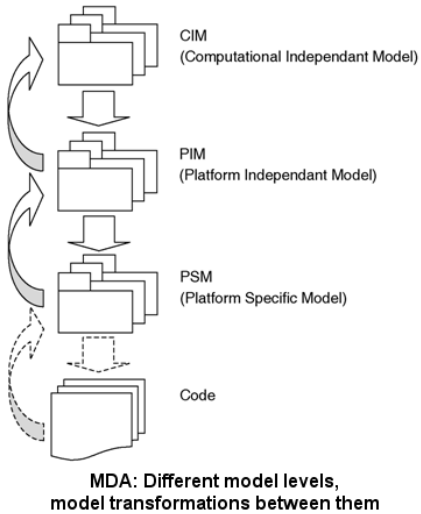

Figure 1.2: The three levels of modeling abstraction in MDA. Figure origin from [16].

The robotics 2020 multi-annual report ICT-2017 highlights the need for the robotics domain to adopt MD and MB techniques to reduce system complexity, enhance reusability and maintainability [6]. Multiple Projects have been funded by the European Union's Horizon 2020 Research and Innovation Program under the RobMoSys project umbrella to develop methods and tools for adopting MD and MB techniques. We refer the reader to the mapping study by Casalaro et al. [12] for a comprehensive overview of MDE in robotic systems.

1.2.2 Behavior Modeling Languages

Throughout this thesis, we investigate the specification of robotics missions using behavior modeling languages. In software engineering, multiple definitions of a model have been proposed, all sharing an emphasis on abstraction [17]. For the purposes of this thesis, a model is defined as a combination taken from Nordmann et al. [11], Baerisch [17], and Casalaro et al. [12]: a model is an abstraction of a system that *"often represents a partial and simplified view of a system or specific aspect"* by *"hid[ing] information about some aspects of the software to present a simpler description of others"*. Crucially, models *"can be processed by sophisticated modeling tools and infrastructures provided by software engineering experts to perform more complex analyses and transformations automatically—thus liberating domain experts from the need to become software engineering experts themselves"*. According to OMG [18], *"structure, terms,*

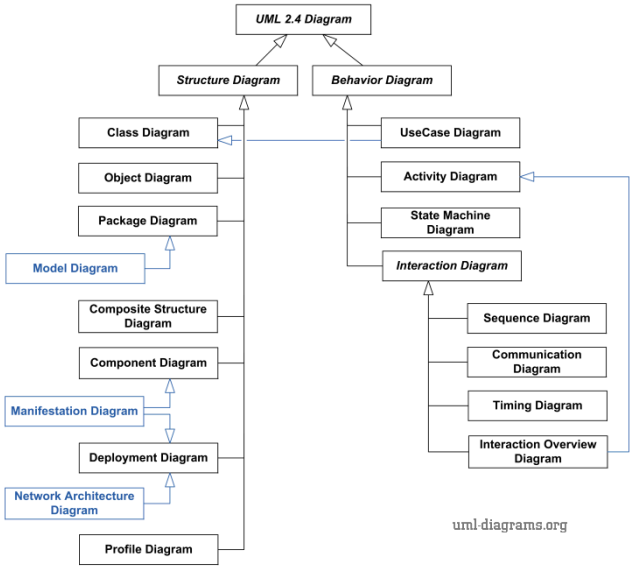

Figure 1.3: An overview of the UML 2.4 classification of diagrams. Figure origin from [19].

notations, syntax, semantics, and integrity rules that are used to express a model constitute the modeling language".

Figure 1.3 provides an overview of the UML classification of models, illustrating various types of behavior modeling languages commonly used in software engineering, noting that additional types may exist beyond those depicted. This classification reflects the UML perspective, and different domains often use different realizations and naming for these languages. For example, state machine diagrams, also referred to as state diagrams and state charts in UML, have a similar concept in robotics called state machines (SMs), also known as finite state machines. However, as our results will show later, they deviate from UML in both the terminology and the concrete realization of this behavior modeling language.

Researchers have advocated for the use of composable behavior-coordination modeling languages, also referred to as behavior modeling languages, to enhance the abstraction level in robotics mission development [5, 20]. In robotics, behavior modeling languages and their supporting tools provide an executable high-level representation of the coordination among different skills that form a mission [21, 22]. By supporting tools, we refer to domain-specific languages (DSLs). DSLs are high-level software languages developed for a specific domain to make it easy for experts to express solutions in that context [23, 24].

In robotics, we could not find a single, widely used term to describe languages or models for specifying robot behavior. For the purposes of this thesis, we adopt the term *behavior modeling language* as a unifying label, even though it is not commonly used in robotics literature. Agent control architectures, execution mechanisms or models for task execution, mathematical models of computation and deliberation technologies are among the terms used in robotics research

and by robotics practitioners to refer to models offering similar functionalities [13, 25, 26, 27, 28]. Behavior trees, state machines, hierarchical state machines, teleo-reactive [25], and decision trees [29] are examples of models used to specify the behavior of robotics missions, each having advantages and disadvantages compared to the others [30, 31]. Throughout the thesis, we use the term *behavior modeling language* to refer to the notation used to create models representing abstractions of robotics missions, and we use the term DSLs to refer to the supporting tools for creating and executing these models. The term is mostly used by practitioners with a software engineering background. With the thesis goal of supporting software engineering practices in the development of robotics missions, we use this term to unify such models.

DSLs are popular solutions for adopting MD and MB in robotics to raise the abstraction level beyond general-purpose languages (GPLs) [11, 12, 32]. The usage of DSLs supports key software practices in robotics, such as separation of concerns [33, 34] and separation of roles [35]. DSLs are categorized as either internal or external DSLs [24]. Internal DSLs are embedded within an existing host language, such as C++ or Python, often as a library. External DSLs are defined as separate languages with distinct syntax, parser, and occasionally textual or visual notation. By offering domain-aligned concepts and notations, DSLs enable users to describe functionalities more effectively [15, 36, 37]. In this thesis, we investigate popular DSLs for two behavior modeling languages in robotics. To support software practices in robotics, we develop our own DSL using MDE techniques to explicitly specify quality concerns in robotics missions.

1.2.3 Behavior Trees

In this thesis, we focus on BTs as an emergent behavior modeling language in robotics. Figure 1.4 provides the visual presentation of the main node types of BTs aggregated from robotics and gaming studies about behavior trees [13, 38, 39]. We illustrate BTs with an example of a disinfecting robotic mission in a healthcare facility in Figure 1.5, shortly explained. In the illustrated scenario, the robot executes a disinfecting sequence across four rooms. For each room, the robot navigates to the location and performs disinfection, repeating this process for all four rooms. If the battery level becomes low, the robot interrupts the disinfecting sequence to recharge.

We use the term robotics missions in the sense of Menghi et al. [22] *"the high-level tasks the robotic software must accomplish"*. Robotic missions typically comprise various skills, often referred to as actions, such as `CollectWayPoints`, which can be programmed at a low level using GPLs. The coordination of these

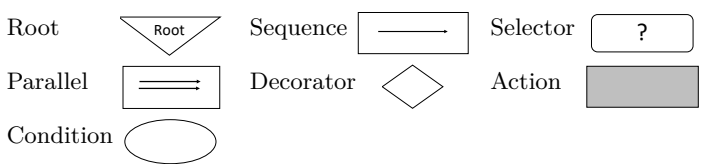

Figure 1.4: Behavior Trees node types (visual syntax) taken from paper A

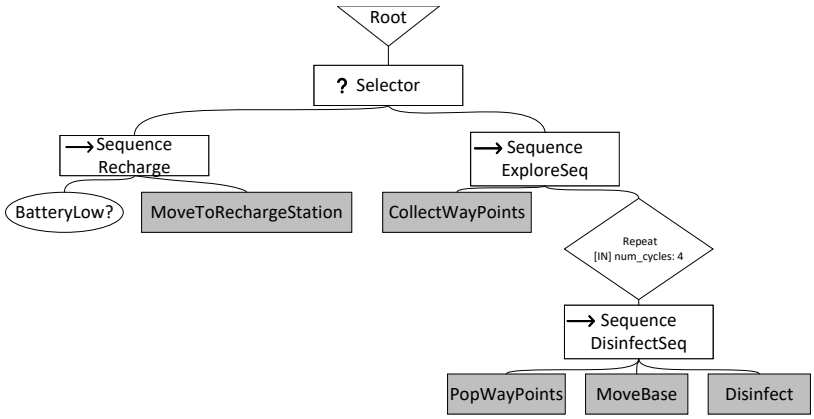

Figure 1.5: An illustration of BTs representing a disinfecting robotic mission. Legend in Figure 1.4.

skills may be implemented at a low level of abstraction in GPLs or modeled at a higher level of abstraction using specialized constructs. For instance, to interrupt the disinfecting sequence for recharging, coordination behavior can be programmed explicitly in a GPL or represented using the `Selector` construct provided by the BT modeling language, as illustrated in Figure 1.5. Employing high-level abstractions for behavior coordination enhances the clarity of the skills involved, beyond implementation details in GPLs, and enables robotic practitioners to modify and maintain robotic missions more efficiently than by altering low-level code.

BTs are directed trees for modeling and executing the coordination of an agent's reactive behavior, with two types of nodes: execution nodes (leaf nodes) and control-flow nodes (non-leaf nodes). Execution nodes represent actions (e.g., `CollectWayPoints`) or conditions (e.g., `BatteryLow`). Main control-flow nodes include sequence, selector, decorator, and parallel. Figure 1.5 shows three sequence nodes (`Recharge`, `ExploreSeq`, `DisinfectSeq`), a selector, and a type of decorators (`Repeat`). BTs operate on ticks, which are signals sent at a fixed frequency from the root to children to update node status: success, failure, or running. For more details, see the background sections in papers A and B. The modularity of BTs allows missions to be broken into reusable tasks called subtrees (e.g., `Recharge` subtree) for flexible, maintainable missions.

In this thesis, we focus on BTs which originated from the gaming domain to model the behavior of autonomous non-player characters [40] and were later used in robotics [13]. They differ from the requirement-specification notation with the same name in behavior engineering [41], which is used to formalize natural-language requirements rather than to model the coordination of robotic behavior. Since this thesis focuses on practical aspects of supporting robotics missions, we do not focus on Dromey [41] requirement-BTs since they are design-time specification models without execution semantics, whereas Colledanchise and Ögren [13] BTs considered in this thesis are used at design time and are executable at run time. In addition, there are different variants of BTs in robotics. We focus on BTs as realized in certain popular robotics DSLs

to provide a relevant comparison with other behavior modeling languages.

Over the past two decades, BTs have become one of the most popular behavior modeling languages for specifying robotic missions because of their modularity, flexibility, and reusability [13, 42, 43, 44]. The growing use of BTs in robotics has led to several DSLs tailored to roboticists' needs [38]. Most existing research focuses on the technical and formal aspects of BTs, as well as advancements in BT formalisms and supporting algorithms [13, 31, 43, 44, 45, 46, 47, 48, 49, 50, 51]. However, empirical studies on BTs from a software engineering perspective remain limited, particularly regarding the tools that support BTs (i.e., their DSLs), including their design and practical use. Many studies are narrow in scope, reflecting only specific practical experience and missing the broader practices and challenges faced by the wider practitioner community. As a result, the current body of knowledge offers only partial insights into the software-engineering dimensions of behavior modeling languages such as BTs. This thesis addresses these gaps by examining BTs through a software engineering lens and exploring additional potential.

1.3 Research Questions

To achieve our research goal, we conducted five studies and organized our work into three main RQs. RQ1 concerns understanding existing support for specifying robotics missions using model-based techniques, with BTs as the primary lens in our investigations. RQ2 builds practical knowledge about BTs adoption in robotics, focusing on open-source projects and practitioners. Drawing on established knowledge, RQ3 addresses the development of solutions to existing gaps in quality concerns and the risk assessment process identified in RQ1 and RQ2. In the following, we list our research questions and the motivation for each.

RQ1. *What support exists for specifying robotics missions using model-based techniques, specifically, behavior modeling languages?*

We seek to understand the existing support of model-based techniques in robotics, particularly the use of behavior modeling languages as a technique to specify robotics missions. We examine the concepts provided by different behavior modeling languages in software engineering and robotics for specifying robotic missions (RQ1.1). We dive deeper into our knowledge building by focusing on BTs, as they currently dominate behavior modeling for robotics missions and lack an empirical understanding in robotics software engineering. We analyze the implementations of BTs in practice by looking into relevant DSLs and compare them to other robotics DSLs for SMs (RQ1.2). We argue that understanding BTs as members of the broader family of behavior modeling languages, and their implementations in practice, contributes to the adoption of more model-based engineering techniques in robotics software engineering.

RQ1.1. *What concepts do behavior trees provide compared to traditional behavior modeling languages in software engineering and robotics?*

We address the current knowledge gap in understanding BTs and their offered concepts as conceptualized in robotics. We position the support provided by BTs against the most well-understood and standardized (via UML) behavior modeling languages in software engineering, activity diagrams and

state diagrams. In addition, we compare the concepts offered by BTs to the traditional and widely used modeling language in robotics, SMs. Reporting and comparing the concepts offered by BTs in practice, as well as against other established behavior models for expressing robotic mission behavior, facilitates a comprehensive understanding of the existing support and gaps for language improvements. Paper A and paper B report the findings of this question.

RQ1.2. *How are behavior-tree DSLs engineered compared to state-machine DSLs in robotics?*

Multiple DSLs were developed to support the implementation of BTs in robotics. However, existing research has not examined the design of these languages from a software engineering perspective, including the support provided compared to established modeling languages in robotics. We investigate the available DSLs for behavior trees and state machines in robotics, analyzing what these languages offer and how they are engineered. By reporting on the current status of the investigated DSLs in robotics, we contribute to building knowledge that informs improvements to these languages and enhance the quality of robotic software systems. Paper A and B reports the findings of this question.

RQ2. *How are behavior trees adopted in robotics software engineering in practice?*

Most available insights about BTs come from theory or polished examples, leaving a knowledge gap regarding practical use and practitioners' practices and experiences with BTs. We want to build this knowledge by empirically investigating how BTs are actually used. By building empirical knowledge, we ensure effective allocation of development and research resources and the creation of support tools for BTs that address practical needs and challenges. We analyze the use of BTs in open-source projects and compare their use with the roboticists' traditional choice, SMs (RQ2.1). Furthermore, we examine, from a software engineering perspective, the experiences of an industrial team adopting BTs through technical action-research and report on both academic and industrial practitioners' practices and experiences using a survey study (RQ2.2).

RQ2.1. *How are behavior trees used compared to state machines in open-source robotic projects?*

Open-source robotics projects offer a window into how BTs are actually used. We analyze open-source projects using behavior-tree and state machine DSLs. We analyze which concepts of the investigated DSLs were used, the characteristics of models in projects, and how BTs and SMs were reused. We compare the DSLs' usage in open-source projects. By comparing BTs and SMs, we ground our observations and help separate what is specific to BTs from what holds more broadly for robotics behavior modeling languages. Paper B reports the findings of this question.

RQ2.2. *What are the experiences and practices of robotics practitioners when adopting behavior trees?*

Our analysis of open-source projects revealed how practitioners use BTs and their DSLs. However, it did not uncover the underlying reasoning, challenges, or workflows behind adoption decisions. Practitioners' own experiences are

needed to surface the practices and workarounds they implement, as well as the perceived benefits and challenges that influence the adoption of BTs. To build this knowledge, we combine a technical action-research study in an industrial context with a survey targeting industrial and academic practitioners to investigate the practices, experiences, and influencing factors involved in adopting BTs. Paper C reports the findings of this question.

RQ3. *How can behavior trees support quality and risk concerns through robotics software engineering?*

Our built knowledge revealed two key gaps. First, robotics missions specified with BTs often lack explicit quality concerns. Second, during the technical action-research study, practitioners highlighted that the risk assessment process and related information are frequently isolated, making integration into robotic mission implementation challenging. Additionally, risk assessment outcomes are typically documented separately from implementation artifacts, leading to incomplete incorporation of essential information.

Both quality concerns and risks assessments information are key concerns in developing reliable robotic missions. In paper A and paper B, we noticed, while analyzing open-source projects using BTs, that even without formal mission descriptions, explicit information modeling within a behavior-tree model helped us comprehend the intended missions, and that this explicit representation facilitated the reuse of tree components. Unfortunately, quality concerns are usually identified late and are not clearly linked to specific mission components because they remain implicit within robotics mission models. Similarly, risk assessment identifies failure modes, and prevention strategies tend to stay isolated from mission implementation in separate models and files. This separation makes it difficult for practitioners to apply and keep track of quality and risk concerns during robotics mission development and implementation. Both types of concern exhibit a common pattern: they are essential to robotic software engineering, are documented in separate files and disconnected from robotics missions implementation, and consequently, are often not implemented consistently. Due to their increasing adoption and role as the coordinating artifact for missions, BTs present a promising boundary object for addressing this gap.

RQ3.1. *How can behavior trees support capturing explicit quality requirements in robotic software systems?*

In robotics, mission specifications define both functional and quality (non-functional) requirements for robotics software systems. Currently, behavior-tree modeling languages make functional requirements explicit by mapping them to action nodes or subtrees, while quality requirements are not directly represented. We want to support capturing quality requirements in the behavior-tree modeling language and make them explicit, ensuring quality concerns are not disconnected from implementation. We propose a meta-model extension for BTs to capture quality requirements. To apply the extension, we develop a fit-for-purpose DSL using model-driven engineering techniques, inspired by observations from our analysis of behavior-tree DSLs. By making quality concerns explicit in a behavior-tree DSL, we ensure that developers can express, refine, and monitor quality requirements within the same model that already executes the robotic mission, supporting a development workflow in which

quality concerns are visible from early design through runtime rather than treated as an afterthought. Paper D reports the findings of this question.

RQ3.2. *How can behavior trees support industrial risk assessment process?*

Risk assessments are important for assuring high-quality robotic missions in industry. Risk assessments processes involve identifying failure modes, prevention strategies, and mitigation that should be transferred into the implementation and execution models of robotics missions. However, practitioners frequently report that conducting risk assessments is time-consuming and that transferring the results is challenging. Although the information produced is essential for safe robotics missions, it typically lives outside the implementation artifact and sometimes reaches developers only partially. In paper C, we observed that the visual and modular structure of BTs, which visualizes decision-making logic, facilitated team communication and improved mission understandability. In addition, we observed that using BTs in the development process led to clear communication of technical details among stakeholders with varying levels of technical expertise. Thus, we investigate how to support the risk assessment process by proposing a model-based approach that centers BTs at the core. By having BTs supporting the risk assessment process and explicitly representing its outputs, we ensure that risk information remains visible from analysis through implementation within the same model used for mission coordination and execution. This approach provides developers with direct access to risk reasoning during implementation and offers safety and risk analysts a living artifact that can be revised as the system evolves. Paper E reports the findings for this question.

Figure 1.6 presents a mapping between the contributing publications to this thesis and the research questions.

1.4 Research Methodology

The development of robotics software systems has grown considerably as a discipline. Yet the state of developing reliable robotic missions using software practices remains only partially understood, with the literature highlighting a limited understanding of current engineering practices and persistent challenges in transferring research solutions into practice [6, 7, 11, 21]. A comprehensive understanding of the field is essential for developing reliable robotic missions and empirically grounded solutions.

This thesis aims to facilitate the development of reliable robotic missions by drawing on software engineering practices. We focus on BTs as one of the dominant mission-coordination models in current robotics practices. Accordingly, we aim to provide empirical insights into BTs in robotics software engineering and to develop empirically grounded solutions by leveraging them.

The nature of this thesis requires a mixed-method approach of knowledge-seeking and solution-seeking studies. In software engineering, knowledge-seeking and solution-seeking studies serve to expand domain knowledge and develop solutions to identified problems [52]. Knowledge-seeking studies strive to understand the field through analysis and observation of software artifacts to reveal problems that require attention, as well as practices and tools that could benefit practitioners and researchers. Once we establish sufficient knowledge of

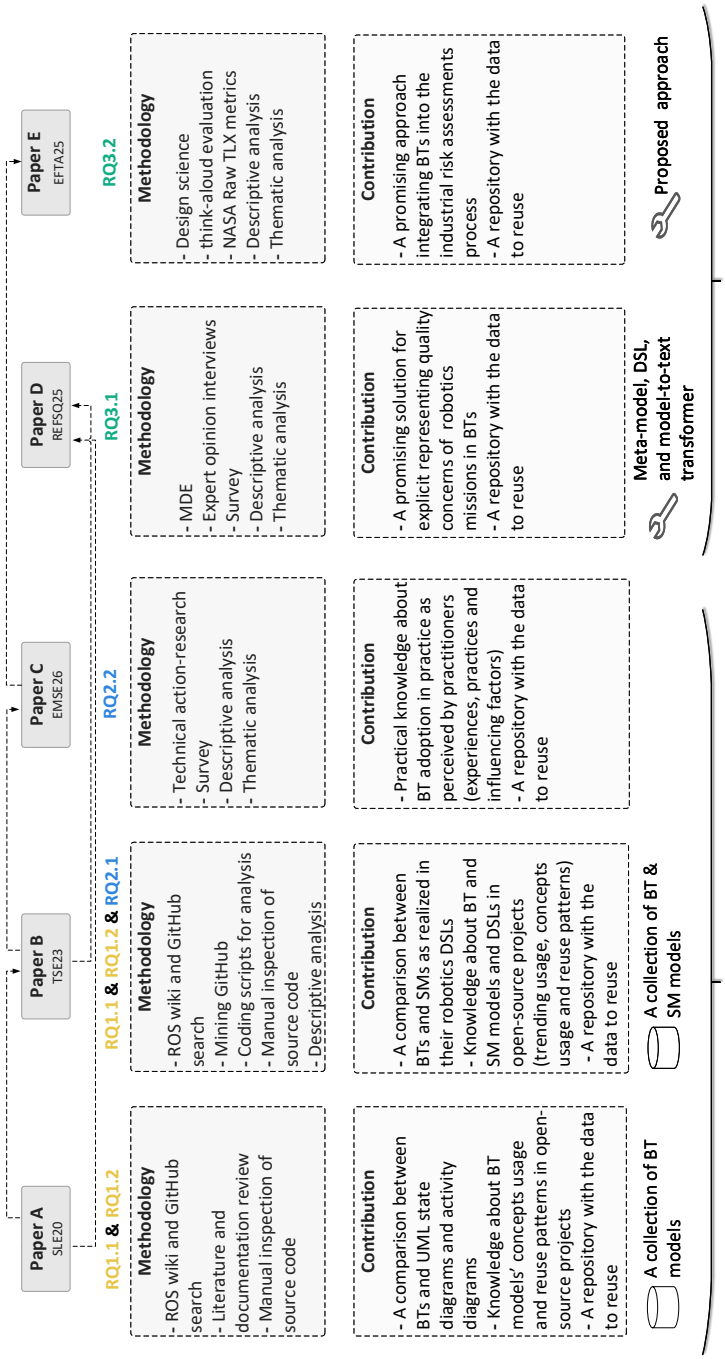

Figure 1.6: An overview of the contributing research papers with their mapping to the RQs.

a software problem, we conduct solution-seeking studies to develop appropriate tools and methods.

We conducted knowledge-seeking studies to answer RQ1 and RQ2 (i.e., papers A-C), and we conducted solution-seeking studies to answer RQ3 (i.e., paper D and paper E). Figure 1.6 provides an overview of the methodologies followed in each study. In the following, we detail the methods used across the constituting studies.

1.4.1 Knowledge-Seeking Studies - RQ1 & RQ2

To facilitate the development of reliable robotic missions, we need to understand the existing support. Thus, we started by building the body of knowledge about existing support for specifying robotic missions using behavior modeling languages in **RQ1**.

In **RQ1.1**, we started with a comprehensive analysis of existing behavior modeling languages in both software engineering and robotics. We focused our lens on the current state of practice for an emerging behavior modeling language in robotics: BTs. We conducted two empirical studies to compare existing support for BTs in comparison to (1) two UML-standardized behavior modeling languages (paper A) and (2) a popular choice among robotic practitioners for modeling robotic behavior (paper B). For the software behavior modeling languages, we chose the most popular, well-understood, and standardized UML behavior modeling languages: state diagrams and activity diagrams. For the robotic behavior modeling language, we chose SMs. SMs are traditional among roboticists and well understood in software engineering, making them a natural choice.

In software engineering, a DSL reflects the concepts and notation that practitioners actually use, making DSL a reliable source for identifying the concepts a behavior modeling language supports in practice. Thus, we searched and identified BT DSLs according to specified criteria on GitHub. We conducted an exploratory literature search and reviewed the documentation of these DSLs to identify existing modeling concepts and their semantics. We mapped identified BT concepts, based on semantics, to the UML behavior modeling languages. We focused on the semantics of BT concepts and examined whether state diagrams and activity diagrams offer direct support for similar concepts or require indirect expression. In addition, we examined characteristics of the behavior modeling languages for supporting users, such as openness of the modeling languages, and users' prerequisites. After understanding the expressiveness of BTs relative to software behavior modeling languages, we shifted the focus of the comparison to the robotic behavior modeling language of SMs. We searched and identified SM DSLs based on the same criteria for BT DSLs using ROS wiki (wiki.ros.org). We used the ROS wiki instead of GitHub because it is commonly used by developers to publish open-source languages and tools for developing robotics applications, making it a reliable source. In addition, a GitHub search for SM DSLs returned hundreds of results with unclear DSL support for robotic systems, rendering the ROS wiki a more consistent data source for our purpose. We reviewed the DSLs related literature and documentation to extract key modeling concepts, their semantics and characteristics of the modeling languages. We then mapped the

BT concepts from paper A to concepts from SM DSLs and analyzed the offered support to users.

Our analysis built an empirical understanding of the offered concepts and modeling support for the investigated behavior modeling languages for specifying robotics missions. To further advance empirical knowledge regarding existing support for robotics behavior modeling, we conducted a comprehensive analysis of the identified DSLs for BTs and SMs (paper B). In **RQ1.2**, we focused the analysis on BTs and SMs, as the RQ1.1 results showed that they provided broader and more adequate support for robotics behavior modeling than their traditional software counterparts, and BT and SM DSLs lack empirical software understanding.

In **RQ1.2**, we examined the identified DSLs from a software language design perspective, focusing on their dynamicity of changing the model, separation of concerns and separation of roles, concrete syntax, concurrency model, implementation type (internal or external) and execution mechanism (interpreted or compiled). We focused on these aspects since they characterize how the modeling languages are realized in their DSLs and how they behave at runtime, shaping users' interaction with the behavior modeling languages. We reviewed tutorials and related publications about the DSLs. We also reviewed the DSLs' GitHub implementations and accompanying documentation. More details can be found in the methodology sections of papers A and B.

So far, our analysis indicated limited empirical understanding of how BTs are adopted in robotics software engineering. The current body of knowledge primarily focuses on the robotic functionality of BTs, such as improving their execution algorithms or integrating them with planners and large language models [13, 49, 53]. There is little empirical evidence on the use of BTs and their DSLs in real projects, and on how practitioners adopt them, leaving a knowledge gap that could inform improvements to current tools and practices. Thus, we decided to dive even deeper into collecting empirical knowledge about the practical adoption of BTs in robotics software engineering in **RQ2**.

In **RQ2.1**, we mined GitHub for open-source projects using the identified DSLs for BTs and SMs. We targeted open-source projects on GitHub since they provide a glimpse into the usage of a concept in practice [54, 55, 56], in our case, BTs. We used a multi-step filtering mechanism after an iterative analysis of the mined projects to exclude projects belonging to a course or tutorial, and to reflect on actual usage of the DSLs. Using the mined projects, we analyzed the popularity of the investigated DSLs over time on GitHub, as popularity trends reveal their adoption trajectories. To conduct a deeper analysis, we sampled the mined projects using random sampling applied to two data pools based on model size (defined as the number of nodes). To capture how practitioners tend to construct models, we calculated metrics that reflect the core structural aspects of the models and reported on the usage of the concepts offered by BT and SM DSLs. To observe how practitioners reuse skills or tasks in BT and SM models, we analyzed the sampled projects using a mix of visual and code-level inspections. We examined reuse patterns across the mined projects, as reuse is a major issue in robotics software engineering [21, 57, 58, 59]. In addition, understanding the support for reuse that a behavior modeling language provides reflects its ability to enhance the maintainability and quality of robotics software [30, 58, 60, 61]. We refer the reader to paper

B methodology section for further details.

In **RQ2.2**, we conducted a two-stage mixed-methods study to build empirical knowledge of practitioners' experiences and practices in adopting BTs. Stage 1 was a technical-action research (TAR) [62] study conducted with a team at an automotive company (company A) to explore and understand the experiences of practitioners adopting BTs in their projects, resulting in a deep understanding of the problem. Stage 2 was a survey study targeting academic and industrial practitioners to gather their experiences and practices, with the aim of enriching and cross-validating our observations from the technical action-research study and achieving a broader understanding.

We adapted Wieringa's TAR structure for our study [62], a practice acceptable in empirical software research [63, 64, 65]. We chose TAR since it is an artifact-driven approach in which researchers actively introduce an artifact into a real-world context to solve a relevant problem and, in doing so, build knowledge about it [62, 66]. We chose the survey approach to collect data because it allowed us to reach a large, diverse group of practitioners from both academia and industry within a manageable timeframe. We direct the reader to paper C methodology section for details about the design of both studies.

1.4.2 Solution-Seeking Studies - RQ3

To facilitate the development of reliable robotic missions, we wanted to build solutions supporting our goal using software engineering practices. The understanding we built of BTs in open-source projects and the TAR study we conducted in an industrial context highlighted two gaps that are key concerns for the development of reliable robotic missions. The first identified gap concerns the lack of support for explicitly specifying quality concerns during the specification of robotics missions using BTs. The second gap came to our attention when an industrial safety expert in the TAR study highlighted that the risk assessment process and information are often isolated and difficult to translate into the implementation of robotics missions. In addition, files and models about the outcome of a risk assessment typically live outside the implementation artifact, making essential information for developing reliable robotic missions partially implemented. **RQ3** investigates developing solutions to support the development of reliable robotic missions by tapping into the two highlighted gaps about quality concerns and risk assessments during our knowledge building phase.

In **RQ3.1**, our goal was to explicitly represent qualities and quality requirements in the implementation of a robotic mission by using BTs. We used an earlier meta-model for BTs that we reverse-engineered in paper A from one of the studied DSLs, and provided an extension for the meta-model with quality concerns (quality and quality requirements). We used a mobile-laboratory robot mission inspired by Burger et al., [67], a mobile robotic chemist, to demonstrate the instantiation of our BT meta-model extension. We designed and developed a DSL to showcase the applicability of our approach and support domain experts. Having the goal of using the best practices from software engineering, we used software language engineering (SLE) and MD practices to develop the DSL. Building upon established solutions, we integrated the DSL into an existing robotics DSL to leverage its functionalities. Finally, we conducted

a preliminary evaluation with six industrial and academic practitioners by holding individual sessions, 20-30 minutes long, with different practitioners and researchers to introduce our solution and share a survey to collect data. We evaluated the feasibility of using the developed DSL to specify quality concerns of robotic missions, and assess the needs, drawbacks, and missing features of the developed DSL. We direct the reader to paper D methodology section for details about the applied MD practices and development of the DSL and the evaluation design.

In **RQ3.2**, our goal was to support practitioners in robotics while conducting risk assessments to overcome challenges they face. We wanted to support the risk assessment process to ensure consistency and traceability between risk assessment findings and their corresponding software implementations by using BTs as a useful boundary object. We followed design science, applying two cycles, to create our solution [68]. We collaborated with an industrial team, a safety expert, and a robotic system developer to develop an approach for using BTs in a risk assessment process. The industrial team was the same as in paper C. We conducted a workshop and multiple meetings with the team to derive relevant requirements for such an approach. Finally, we evaluated the approach with five practitioners from different companies using a think-aloud study [69]. We conducted a recorded one-hour session per practitioner. We assessed the effectiveness of using BTs in the process of risk assessment, focusing on risk identification, integration of outputs into the implementation phase, and visualization of results. We analyzed the data using thematic analysis [70]. We direct the reader to paper E methodology section for further details about the development process of the approach.

1.5 Results & Discussion

In this section, we revisit the research questions and provide corresponding answers. Additionally, where applicable, we reflect on the research impact of the contributions and appended papers. We refer the reader to Figure 1.6 for an overview of the contributions in each study and the mapping to the RQs.

1.5.1 Existing Support Using Behavior Modeling Languages (RQ1)

In the following, we summarize our results describing existing support for specifying robotics missions using different robotics and software behavior modeling languages. In addition, we summarize results that describe the software language engineering behind the investigated behavior modeling languages' DSLs.

1.5.1.1 Concepts Offered by Robotics Behavior Trees and Other Behavior Modeling Languages (RQ1.1)

RQ1.1 is answered through our empirical research in paper A and paper B. In the papers, we extracted offered concepts from BT relevant literature and DSLs, `BehaviorTree.CPP` and `PyTrees` family (`PyTrees` and its ROS extension `PyTrees_ros`). Furthermore, we extracted offered concepts from SM relevant literature and DSLs, `SMACH` and `FlexBe`. Finally, we compared the concepts

extracted from BTs with those extracted from SMs and the two UML behavior modeling languages, state diagrams and state machines. We refer the reader to papers A and B for details on the results.

Our analysis in RQ1.1 showed that BT and SM languages provided a wider range of concepts for coordinating behavior using design patterns than the investigated UML behavior modeling languages. Table 1.1 shows an overview of the concepts offered by the investigated behavior modeling languages. The leftmost column lists concepts relevant to BTs due to inclusion in BT DSLs. The last column briefly comments on how the respective concepts are handled in SMs, as realized in the DSLs and the two UML behavior modeling languages. As presented in Table 1.1, the investigated UML behavior modeling languages required customization to offer similar support. This finding is not unique to the UML modeling languages investigated. In general, the need to customize UML modeling languages before using them in practice is a common feature [32, 71]. It might be related to concerns about making a modeling language complex if design patterns are implemented explicitly as constructs in it. However, the need to customize UML modeling languages seems to deter robotics practitioners. Robotics practitioners prefer to use already-customized modeling languages, such as DSLs, over UML-based ones due to their expressiveness and ease of use [32].

Openness of the robotics modeling languages, BTs and SMs, was a first-class concern, supported by both of them and expected by the users. In both BT and SM languages, the ability to extend the languages' semantics beyond available concepts was possible. Only bare-bones functionalities were provided in the meta-classes of simple node types in BTs and basic state types in SMs, and users were expected to extend them. The same was not true for the UML languages investigated. UML's State diagrams and activity diagrams did not openly support the extension of their concepts nor require the users to do so. Of course, it was possible to extend the UML modeling languages to support a wider set of concepts by using advanced modeling techniques, such as stereotyping [72]. However, it required advanced knowledge of modeling techniques and was seen rather rarely.

We should recall that the examined BT and SM DSLs emerged from collaborations between academic and industrial practitioners to address specific challenges [13, 14, 73, 74]. Consequently, openness was supported in both robotics behavior modeling languages through extensible meta-models in response to roboticists' needs. The need for openness of the robotics behavior modeling languages comes from the need to easily adapt to diverse scenarios in robotics and the lack of agreement between robotics practitioners about the ideal control model for robot behavior. Although robotics practitioners are usually not trained in meta-programming, this language design seems needed and well received in light of the DSLs usage trend later discussed in RQ2.1 and RQ2.2 (check Fig. 3.8 for open-source usage and Fig. 4.7 for usage as reported by practitioners).

Table 1.1: Selected key model concepts for BTs compared with robotics SMs and two UML diagrams. Content originates from combining two tables in paper A and paper B.

Concept	**BT languages** `PyTrees`, `PyTrees_ros`, `BehaviorTree.CPP`	**SM languages** `SMACH`, `FlexBe`	**Activity diagrams**	**State diagrams**
simple nodes	Execute **actions** (arbitrary commands, both instantaneous and long-lasting) or evaluate **conditions** (value translated to success/failure)	Basic state with associated action/conditions	Basic activity	Basic action
exit status	Each node reports success, failure, or an in-operation state ("running") each time it is triggered. Status report causes the computation (the traversal) to advance to the next node	Each state reports its status (also known as outcome). Outcomes are user defined. Execute function checks periodically for outcome	Completion of an activity advances state like in BTs, failures modeled by exceptions/handlers	No direct support, control-flow mostly driven by message passing, not by activity flow
composite nodes	Define hierarchical traversal, the control-flow for each epoch (tick). Sequentially composed. Nodes may start concurrent code though	Similar to container concept. Allow state nesting and control flow constructs	Nested activities	Nested states, both sequential and parallel
root	Serves as entry point for every traversal. Has exactly one child node. The root node is re-entered at every epoch	Initial state (first state added to the container)	Initial node, possibly a root activity, initialized only once	Initial state, possibly a root state, initialized only once
sequence	Trigger children in a sequence until the first failure. If no failure return success, otherwise fail	Supported using a predefined container	No direct support, detect failure with exception handler	No direct support, use a parent transition on failure
selector	Trigger children in a sequence until the first success. If no success return failure, otherwise succeed	No direct support, could use container to extend.	No direct support, difficult to model with exceptions	No direct support, use a parent transition on success

Table 1.1: Concepts comparison (continued)

Concept	**BT languages** `PyTrees`, `PyTrees_ros`, `BehaviorTree.CPP`	**SM languages** `SMACH`, `FlexBe`	**Activity diagrams**	**State diagrams**
parallel	Generalize sequence/selector with a policy parameter. Several polices available, e.g. meeting a minimum number succeeding children	No direct support, could use guards to extend	No direct support	No direct support
goto (jumps)	No general jump construct, the computation always traverses the tree	Supported	Supported	Supported
decorators				
inverter	Invert the Success/Failure status of the child	No direct support, could use container to extend	No expl. inversion operator	No expl. inversion operator
repeat	Trigger the child node a set number of times, then succeed. Fail if the child fails	No direct support. Could use the *Iterator container* in `SMACH` as repeat-until loop where the desired outcome for breaking the loop is user-defined	No direct support. Encode with loops + counter/-guard	No direct support. Encode with loops + counter/-guard
succeed	Return success regardless of status returned by the child	No direct support, could use container to extend	No expl. support for status	No expl. support for status
retry	Run the child node and retry it immediately if it fails for a maximum number of times, otherwise succeed	No direct support. Similar to repeat concept, *Iterator container* can be modified and used	No direct support. Encode with loops + counter/-guard	No direct support. Encode with loops + counter/-guard

1.5.1.2 The Engineering Behind Behavior-Tree and State-Machine DSLs (RQ1.2)

RQ1.2 is answered as part of paper B findings. In the paper, we analyzed the software languages design of five DSLs. For BT DSLs, we analyzed the C++ library `BehaviorTree.CPP`, the Python library `PyTrees` and `PyTrees_ros`, an extension for `PyTrees` to support ROS binding. Throughout the thesis, we are going to use the term `PyTrees` family to refer to both `PyTrees` and `PyTrees_ros` since they share the same language design with modifications to support ROS. For SM DSLs, we analyzed the two Python libraries `SMACH` and `FlexBe`. Our analysis focused on implementation type (internal or external DSL), dynamic modification capabilities, programming model, execution mechanism (interpreted or compiled), support for visualization (GUI support), and finally separation of concerns and separation of roles. We refer the reader to paper B for details about the results.

Our findings highlighted similarities and differences in the design of the DSL. Dynamic modification capabilities, interpreted execution models, and the ability to specify concurrency models were among the common features of the investigated DSLs. Compared to the investigated UML languages, the implementation of the UML languages did not usually allow altering the execution order during runtime or offer comparable features. Robotics missions inherently demand continuous adaptation to dynamic environments and evolving tasks. Furthermore, robotics software systems often integrate different hardware and software components, necessitating customization to meet user requirements. Consequently, incorporating such features into the design of robotics DSLs is a more pressing concern than in the domains where UML-based languages traditionally originated. In terms of differences, we observed that `BehaviorTree.CPP` and `FlexBe` could be clustered into one design group, and the `PyTrees` family and `SMACH` into another. Adapting specific MD techniques, providing a GUI, supporting separation of roles and supporting model consistency verification were among the features that distinguished the two groups of DSLs.

We observed similarities and differences among the investigated DSLs, which shaped how practitioners specified robotics missions in the models examined later in RQ2.1. As we discuss later in RQ2.1, certain design decisions affected the usage of certain concepts and our ability to analyze sampled models. One of the design decisions was the close coupling of models to specific robotic platforms across all DSLs, which hinders reuse and external analysis. Only `BehaviorTree.CPP` allowed model analysis without a working robotic environment. Analyzing models from the other DSLs in RQ2.1 required us to implement workarounds. In an empirical study targeting companies and industrial practitioners, decoupling robotic systems was reported as an important aspect that companies strive to support to promote reusable robotic systems across projects, and the researchers highlighted the need for tools promoting software-hardware decoupling of software components [75]. For robotics missions development, we recommend the same to language developers to achieve a clearer separation and decoupling between models and the execution environment, and we anticipate advancement in the supporting DSLs. Enhanced separation would facilitate external processing for visualization, testing, and

reuse, thereby improving interoperability, productivity, and portability of robotic systems [32].

Reflecting on both RQ1.1 and RQ1.2 findings, we observed that adapting modeling and language-engineering methods was relevant in practice and well received by robotics practitioners. The investigated DSLs drew on modeling and language engineering practices, adapting them well to the needs of domain users. Robotics practitioners come from lower-level programming paradigms. The well-designed DSLs allowed them to raise the level of abstraction and use the robotics behavior modeling languages, BTs and SMs, to effectively implement missions of robots in higher-level representations. This is a clear case of using software practices and redefining them to adapt to the needs of other disciplines. Researchers in software engineering have been pushing to do that, especially for adapting abstraction engineering and MDE techniques within other disciplines [76, 77]. We need to remember that all the investigated DSLs emerged from practical needs of roboticists (bottom-up language design). So, other developers of robotics languages would benefit from using the available knowledge in this thesis about their design and usage patterns. As stated by de Araújo Silva et al. [32] "*The most successful MDE practice is driven from the ground up*".

Recommendations inspired by RQ1 findings

Concepts and modeling openess. We recommend using our reported data on supported concepts and on the openness of behavior modeling languages in robotics to inform the development of future modeling languages, keeping in mind that the reported features reflect practitioners' needs.

DSL design features needed by Roboticist. We recommend using the reported similarities as starting guidelines for robotics language designers since incorporating such features is needed by practitioners.

From the ground up. We recommend using the analysis of robotics behavior modeling languages and their DSLs as inspiration for adapting more modeling and language-engineering practices, such as MD techniques.

1.5.2 The Adoption of Behavior Trees in Practice (RQ2)

In the following, we summarize results on the adoption of BTs in practice. We summarize the results from comparing the usage of BTs and SMs in open-source projects. Finally, we summarize the experiences and practices followed by robotics practitioners when using BTs in their projects.

1.5.2.1 Behavior Trees and State Machines in Open-Source Projects (RQ2.1)

RQ2.1 is answered as part of the findings in paper B. We mined from GitHub[1] 1086 projects using `SMACH` and `FlexBe` and 271 projects using `BehaviorTree.CPP` and `PyTrees` family. After applying our filtering mechanism to discard course and tutorial projects, we had 620 projects using SM DSLs,

[1] the mining was conducted until 31/12/2021

including 2507 SM models, and 169 projects using BT DSLs, including 658 BT models. Our random sampling of BT and SM models in the mined projects yielded a total of 150 models (75 BT models and 75 SM models), belonging to 43 projects (25 BT projects and 18 SM projects) from eleven domains (check Figure 3.9). We analyzed the usage trend of the investigated DSLs in open-source projects and how practitioners constructed BT and SM models by examining metrics that capture core structural aspects of the models and the usage of the DSLs' offered concepts in the models. Finally, we reported three patterns of model reuse among users. We refer the reader to paper B for details about the results.

The metrics we calculated showed that practitioners kept the overall model structure simple in both behavior modeling languages (shallow models of moderate size). Yet, our metrics revealed differences in how practitioners built BT and SM models, particularly in the size and vertical structure of both models (depth for BTs and nesting level for SMs). For example, BT models had an average model size three times that of SM models across the sampled population. The practitioners' preference for larger BT models may be attributed to the explicit modeling of control-flow constructs in the BT meta-model. With moderate BTs size, readability is generally maintained [31]. In contrast, the typical SM meta-model does not enforce explicit control-flow constructs, which may influence practitioners' modeling habits and result in smaller models. Furthermore, it is well established that as SMs grow in size, their readability is affected negatively [31, 48].

Connecting our findings in RQ2.1 to the RQ1.2 findings on the DSLs' design revealed that the way models were constructed and the usage patterns of concepts were not incidental. We observed that certain usage patterns might be traced back to specific design decisions within the DSLs. We observed distinctions in the calculated metrics among the DSLs within the same behavior modeling language that might relate to GUI support and underlying DSL development language (C++ and Python). `FlexBe` and `BehaviorTree.CPP` shared language design characteristics that distinguished them from `SMACH` and the `PyTrees` family. Both `FlexBe` and `BehaviorTree.CPP` adopted specific MD techniques, provided a GUI, and supported model consistency verification. Interestingly, both 'DSLs' usage growth in open-source projects was much higher than that of `SMACH` and the `PyTrees` family (check Figure 3.8). Although we cannot confirm whether that is the reason for their popularity relative to the other two DSLs, it would be interesting to investigate further. Additionally, we observed that those design decisions might also have influenced the use of DSL concepts and the reuse of models. Notably, certain built-in constructs were used more frequently in `FlexBe` and `BehaviorTree.CPP`, which offered GUI support and are external DSLs. Whereas projects using `SMACH` and the `PyTrees` family tended to embed such constructs directly in code rather than in the model, noting that both are internal DSLs without GUI support. This pattern suggests that external DSLs with GUI support promoted consistent use of DSL constructs, whereas users of internal DSLs are more likely to rely on GPL constructs. Furthermore, we experienced a harder time analyzing models that integrated constructs directly into code, rather than abstracting them into models. This insight is relevant for both robotics DSL developers designing future behavior-modeling DSLs and practitioners selecting among available

options.

We observed the choice of reuse mechanism was influenced by the design of the DSL, whether one reuses skills or tasks and the available support of predefined skills to reuse them. In our work, reusing refers to the ability to reuse skill code (also known as an action) or reuse code of a repeated task (composed of different skills) in the same model or across models in a project. We use skill-level and task-level to reference each concept, respectively. Our analysis showed that predefined skills APIs could facilitate reuse. For example, `ROS Navigation Stack skills` (wiki.ros.org/navigation) was used in 55% of the `SMACH` projects and 60% of the `PyTrees_ros` projects. The same was observed in `FlexBe` offered skills APIs and supporting libraries where the predefined skills were used in 70% of projects. This observation suggests potential to support additional predefined skills APIs and mechanisms and could be worth considering by developers to facilitate reuse. As stated by Bencomo et al [76]: *The ability to reuse existing components in new contexts is essential to the effective development of software systems. It is an important engineering principle and a sign of a maturity of a discipline.*

Finally, we provide an open-source repository containing 2507 SM models and 658 BT models to facilitate future research. To ensure transparency and reproducibility, we include all code for model analysis and visualization, as well as calculated metrics organized in Excel sheets for further analysis. We encourage researchers to build upon our work.

1.5.2.2 The Experiences and Practices adopting Behavior Trees by Robotic Practitioners (RQ2.2)

RQ2.2 is answered in paper C. In the paper, we conducted a TAR study with industrial practitioners at Company A, and a survey study with 34 industrial and academic robotics practitioners. The results covered BTs adoption by robotic practitioners exposing practices, experiences and influencing factors for different software engineering aspects. The paper findings covered eight interest areas spanning BT design, integration, and use in practice. The areas of interest were identified during our TAR study on the adoption of BTs with the industrial practitioners. The eight interest areas were: granularity of BT model, BT libraries, BT-to-ROS nodes architectural design, mixing programming languages during the programming of BTs, team communication, understandability, the usage of planning algorithms with BTs and the scalability of BTs. We provided several directions for both researchers and practitioners as implications to certain findings. We refer the reader to paper C for more details.

For context, we briefly introduce information about the survey respondents and the industrial practitioners at Company A. Most survey respondents were from industry, comprising 67.6% of the total respondents, while academic respondents accounted for 29.4%. Most respondents reported over 2 years of experience in robotics (82% of total responses), and only 18% of respondents had less than 2 years of experience. The majority of respondents used BTs in their projects (91% of total respondents), and only a minor percentage used BTs in toy and tutorial contexts (38% of total respondents) (check Figure 4.3). Regarding the TAR study participants, we had three participants; two of them

were actively involved throughout the cycles. The two participants were a safety expert and an industrial researcher. None of the participants had prior experience with BTs. In the following, we use the term practitioners when referring to both survey respondents and TAR participants; otherwise, we refer to them as respondents and participants, respectively. We also use the term library instead of DSL when referring to supporting tools for using the behavior modeling language, BTs.

An interesting connection to our findings in RQ1.2 and RQ2.1 is that practitioners reported positive experiences and implementation challenges that were related to BT libraries' design decisions, `BehaviorTree.CPP` and the `PyTrees` family. Improvements in providing practical guidelines for starting with BTs, integrating them with ROS, and developing additional supporting tools were among our findings. For example, practitioners preferred using `BehaviorTree.CPP` over other BT libraries (check Figure 4.7). They reported that it was easy to start with `BehaviorTree.CPP` because the GUI was decoupled from the implementation, making it easy to begin modeling and visualizing a robotic mission. We reported in RQ1.2 and RQ2.1 how the coupling of models to a specific platform and requiring a working environment to start modeling and visualizing a robotic mission, except for `BehaviorTree.CPP`, would hinder practitioners' usage of a library. In general, the reported findings further emphasize how a library design choice affects practitioners' usage and experience, and the need to build empirical knowledge by talking to practitioners to develop solutions from the ground up.

Beyond implementation, we examined practitioners' experiences concerning the impact of BTs on team communication, as well as their influence on understanding decision-making logic and the execution of robotic missions. In general, practitioners indicated that BTs facilitated improved communication among team members and non-technical stakeholders about technical details. In addition, BTs enhanced the clarity of robotic decision-making logic. This finding concerns the abstraction of behavioral coordination and the composition of tasks for a robotic mission using a behavior modeling language, in this case BTs, and underscores the need for supporting tools to enable this process [78]. Clear communication among different types of stakeholders and a good understanding of the decision-making logic of robotic missions are important for the development of reliable robotic missions and the success of robotic systems in industry [75].

Looking at the overall experience reported by practitioners, specifically certain reported challenges about documentation gaps and transitioning from BTs modeling to integration with ROS, a recurring observation draws our attention. The TAR participants who were new to BTs found certain aspects challenging, whereas survey respondents with BT experience did not collectively report them as challenging. This disparity may be attributed to an initial learning curve faced by new users, which diminishes over time (onboarding cost). Additionally, experienced users may have established their own workarounds, so they no longer notice these gaps and challenges. In RQ2.1, we created our own workaround to analyze models that lacked GUI support in their DSLs, suggesting that other practitioners employ similar practices. In empirical software engineering research, this friction between new and experienced users is not new, and experienced practitioners tend to compensate for inadequate

support by devising workarounds [79, 80].

Reflecting on practitioners' reported experiences and practices, we provided in paper C directions for practitioners and researchers to reinterpret the findings as a future agenda rather than mere reporting. The majority of the implications fall into improvements to certain supporting algorithms and libraries, as well as the development of solutions that incorporate reported experiences and relevant contextual factors. Additionally, we observed several challenges and areas for improvement that emerged from the broader context in which BTs were applied, rather than from the formalism of BTs alone. We foresee a shift in the research agenda for further empirical research to understand how BTs and other behavior modeling languages are used effectively by treating context as a primary variable, i.e., what works for whom, where, when, and why [81, 82]. To enable practitioners to develop reliable robotics missions by adapting abstraction and MD techniques, it is essential to consider the statement by Whittle et al. [71] regarding the success of MDE techniques in industry: *[...] social and organizational factors are at least as important in determining success as technical ones.*

Finally, we provide an open-source repository containing the survey instrument, the anonymized answers and the thematic analysis coding as a replication package to support the transparency of research and to support further research.

Recommendations inspired by RQ2 findings

Intentional design decisions. We recommend treating design choices such as internal vs external realization, user flexibility, and graphical support as intentional design decisions rather than technical afterthoughts, since these factors influence how practitioners construct, experience, and reuse their models.

Offered concepts and reuse. We recommend using our analysis data in open-source projects focused on concept usage and reuse to guide improvements in current BT DSLs and their successors.

Starting points for improvements. We recommend using our reported findings about practitioners' experiences, practices and influencing factors when adopting BTs to guide further efforts for improvements.

Avoiding a one-size-fits-all We recommend avoiding a one-size-fits-all approach and instead prioritizing contextual factors, such as target users, as primary variables when developing supporting methods, algorithms, and tools for BTs and other behavior modeling languages.

1.5.3 Supporting Quality and Risk Concerns Using Behavior Trees (RQ3)

In the following section, we summarize results on integrating BTs with software engineering practices to enhance the reliability of robotic missions. This approach specifically addresses gaps in explicit specification of quality concerns, as well as supporting the risk assessments process identified during the

knowledge-building phase in RQ1 and RQ2.

1.5.3.1 A Solution for Capturing Quality Requirements Using Behavior Trees (RQ3.1)

RQ3.1 is answered in paper D. We developed a solution to explicitly specify quality concerns in a robotic mission by incorporating them into the behavior modeling language, BTs, using SLE and MD techniques from software engineering. Our solution consists of three parts: a quality-focused meta-model extension of BTs, a textual DSL to explicitly express quality concerns within BTs, and a model-to-text transformation to transform any model created with our DSL into a `BehaviorTree.CPP` XML-like code format. We direct the reader to paper D for details about the solution and evaluation results with external practitioners. In this context, we discuss the contributions of our solution and the implications of the solution's evaluation results.

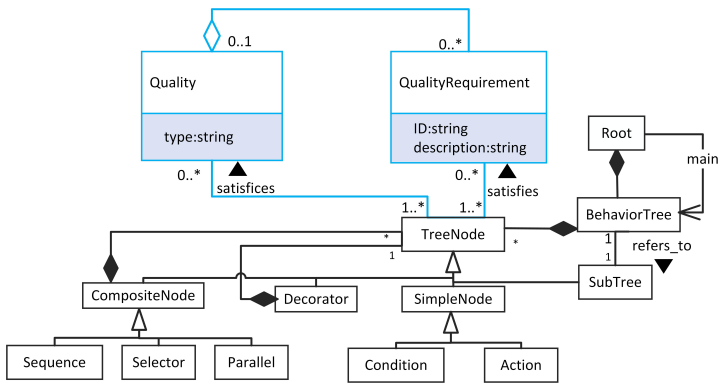

Figure 1.7: Our quality-focused meta-model extension of BT modeling language from paper D

Our approach paves the way for the explicit incorporation of quality concerns into robotics mission behavior modeling languages. This promotes early expression of qualities in robotics missions and provides a clearer overview of mission components and their contributions to addressing quality concerns. When a desired behavior of a robotic system cannot be explicitly specified and traced to specific components, it is insufficient to claim that the validity of requirements alone guarantees the system's reliability [83]. Figure 1.7 presents the first part of our solution, the quality-focused meta-model of BTs. We designed the meta-model to be generally applicable to all kinds of quality requirements and provide flexibility to domain experts throughout the lifecycle of a mission. By lifecycle of the mission, we refer to early and later design time, as well as the implementation (coding) of the mission.

Figure 1.8 shows an overview of the user process to use the textual DSL and a snip of the DSL syntax. By using insights about DSL design and usage from paper A and paper B (RQ1 and RQ2), we contributed a textual DSL with an intuitive and light syntax taken from the literature. The idea behind providing the model-to-text transformation is to leverage existing functionalities

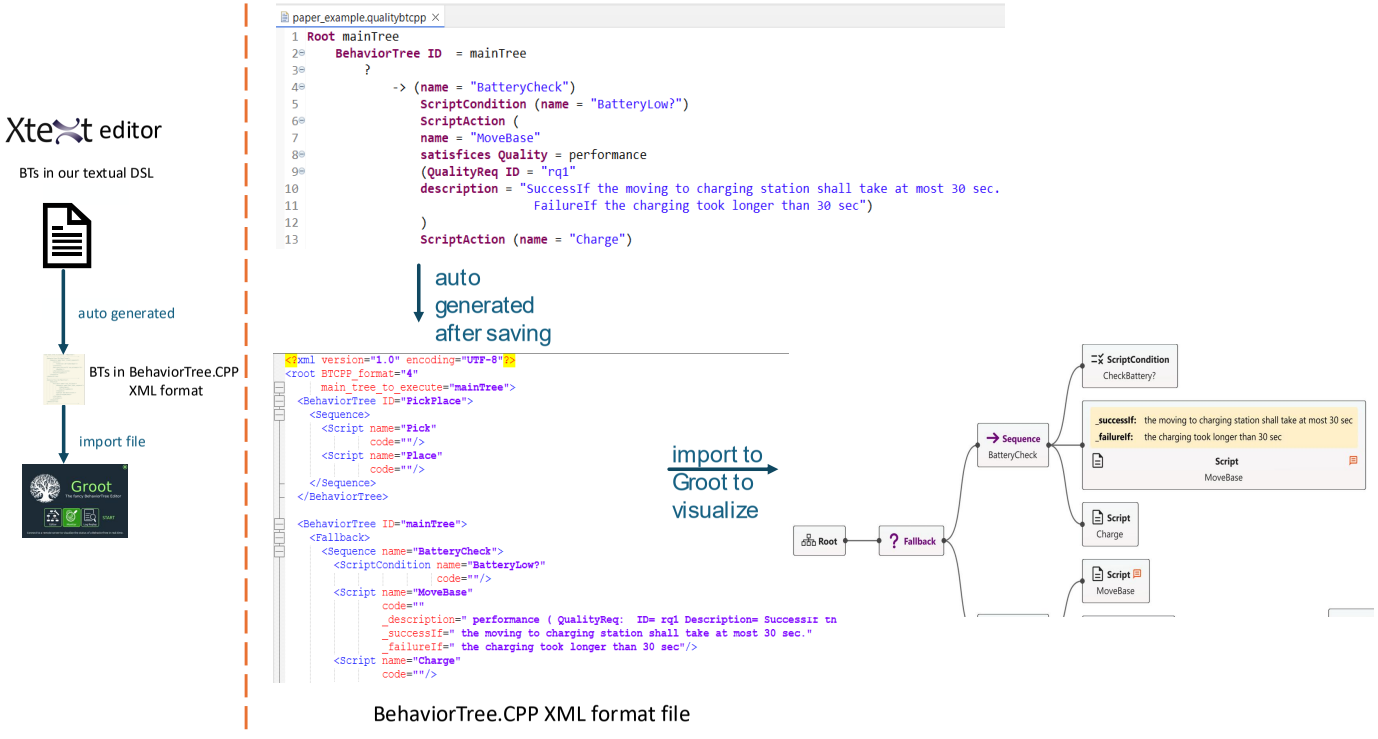

Figure 1.8: An overview of the user process to use the DSL

in `BehaviorTree.CPP`, which RQ2 highlighted the most preferred BT library by practitioners and open-source users. In addition, we observed in paper B the effect of having a GUI on the explicit usage of concepts; thus, we wanted to have a graphical visualization to further support practitioners in specifying quality concerns explicitly. With the integration to `BehaviorTree.CPP`, we support users in using its GUI, Groot, to visualize BTs with quality concerns and we do not disrupt users' existing workflows.

The results of our preliminary evaluation were promising and provided insights into drawbacks in the DSL design and potential feature enhancements. All participants expressed the benefit of using our DSL to express qualities directly in BTs at early-stage design time of robotic missions, and they appreciated the overview it provided for important qualities in BTs. Two drawbacks were expressed about the usage of Eclipse and the DSL being a textual DSL instead of a graphical one. Three features were requested for future enhancements regarding automatic runtime monitoring for quality compliance, providing the probability of quality compliance, and an easy way for reuse of cross-cutting quality requirements.

In software requirements engineering, quality concerns have long been recognized as a key factor in system success [84]. Moreover, standardization bodies such as the IEEE Robotics and Automation Society (RAS) are developing guidelines, with one recommendation being to develop better tools and methods for explicit requirements elicitation and verification, as today's practices fall short [83, 85]. Reflecting on our evaluation results and the general state of research on comparable solutions indicates a need among practitioners for similar solutions that enable the explicit specification of quality concerns at the earliest stages of robotics missions [21, 86, 87]. We emphasize the need to support existing practitioners' workflows rather than disrupt them and provide a flexible and iterative way to specify quality concerns. We also emphasize leveraging existing knowledge in this thesis about existing support and reported experiences, so that researchers do not duplicate effort.

Finally, we provide an online repository for all the solution parts. We hope others build on top of our solution and create their own with the goal of making quality concerns explicit for reliable robotic missions.

1.5.3.2 A Solution for Supporting Industrial Risk Assessment Process Using Behavior Trees (RQ3.2)

RQ3.2 is answered in paper E. Our solution consists of two parts: a list of five derived requirements for designing our approach, and our approach, which supports early risk assessment in a development-centric way. The solution is the result of collaboration with the industrial team from RQ2.2, during which we spent time deriving the requirements and developing the approach. We direct the reader to paper E for details about the solution and evaluation results with external practitioners. In this context, we discuss the contributions of our solution and the implications of the solution's evaluation results.

The derived requirements spanned multiple aspects that targeted supporting safety experts with no prior experience and identifying risks as early as possible during the risk assessment process (support an early and iterative process, support low-dependency on prior data, and support visualization of mission

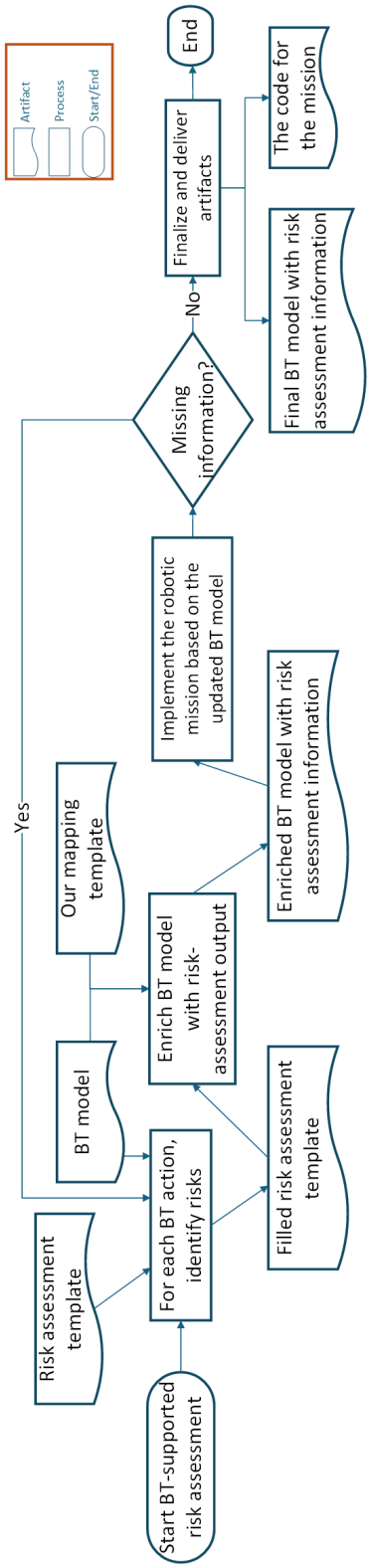

Figure 1.9: Our approach for supporting risk assessment with BTs from paper E

behavior). Furthermore, the derived requirements required integrating the risk assessment findings into the life cycle of missions development (support tracking in code, support light and accessible documentation). We suggest that the derived requirements may hold for other companies, since the needs that gave rise to the requirements stem from companies racing to adopt new intelligent and collaborative robotics systems, especially with the growth of Industry 5.0 [88], and from regulatory bodies seeking to ensure reliable and safe robotics missions [83].

Figure 1.9 presents an overview of the proposed approach, which integrates BTs to support the risk assessment process. We aimed to integrate risk-assessment findings into the engineering workflow rather than maintain them in a separate silo by employing BTs as a boundary object. BTs are already utilized by practitioners throughout the entire development cycle of robotics missions. Therefore, incorporating them early in the risk assessment process, rather than relying on Excel-based documentation and outcomes transfer, is a logical progression. As Bucchiarone et al. [77] state: "Models in model-based systems engineering allow for a more systematic representation of information compared to documents, such as spreadsheets".

Our evaluation results with external industry practitioners yielded positive aspects, such as BTs as a mental model for risk identification. It also yielded future seeds for further development, such as providing version control for change traceability and suggesting an extension to an existing BT library to support the visualization of risk assessment information. By leveraging BTs' strength in clearly visualizing the behavioral logic of a mission, they could serve as an information bridge to ensure alignment between high-level risk assessments and low-level implementation (code).

The overall findings in RQ3.2 indicate that BTs can serve as a shared behavior-modeling language spanning the robotics mission development cycle, from early risk identification to implementation. This positions BTs as boundary objects between cybersecurity engineering and robotics software engineering practices, a role that introduces several open research questions. A potential open research question comes from one of our evaluation results. The result highlighted that the level of granularity of the BT model used in the process can both clarify the mission and introduce bias in hazard identification, so it is a challenging decision to take. This challenge is not solely for using BTs in the risk assessment process. One of the findings in RQ2.2 showed, in general, how practitioners find it challenging to determine the appropriate level of granularity to model in BTs. Further investigation is needed into how early modeling decisions, such as the chosen level of granularity, affect the results of a risk assessment process. Another open question comes from the lack of available tooling for supporting BTs representing risk assessments outcomes. In paper E, we used `BehaviorTree.CPP` GUI, Groot to represent risk assessments outcomes in BTs and assess the approach. Our evaluation highlighted the need for better tooling, which is normal since Groot is not designed for this purpose in mind. However, this finding highlights how the current BT libraries and editors lack consideration for risk information. A promising research direction could investigate further potential risk-aware BT libraries. Finally, we do not claim that our approach is generalizable. Further empirical studies involving practitioners are necessary to determine whether BT-centric risk assessment

can scale beyond individual missions and generalize across domains.

Recommendations inspired by RQ3 findings

Seeds for further development. We recommend leveraging the evaluation results for the developed DSL and the online materials provided in RQ3.1 to advance the solution, as the current solution demonstrated potential according to the evaluation responses.

Introducing open questions. The use of BTs as boundary objects between two domains demonstrated potential and opened further questions. We recommend that researchers explore additional applications of behavior modeling languages, such as BTs, to bridge gaps among practitioner groups.

A new research direction. Reflecting on the findings of RQ3.1 and RQ3.2, we foresee a research direction toward combining both solutions to better support robotics practitioners throughout the entire cycle of developing robotics missions.

1.6 Research Limitations

In this section, we consider the thesis as a whole, evaluating the scope of our combined evidence and the extent of our findings. The appended papers address the threats to validity associated with each individual study, where we direct the reader for more information.

Our knowledge-seeking findings are derived from specific variants and literature of BTs, open-source projects and practitioners. The derived practitioners' knowledge comes from a community that has already adopted BTs in ROS ecosystem and new users in the same settings. Consequently, our research focuses on creating knowledge about BTs in specific community context, rather than the broader robotics community. For example in RQ1.2 and RQ2.1, we highlighted that DSL design decisions, such as having a GUI support and the coupling of models to specific platforms, influence how practitioners construct and reuse their models. Our conclusions about concept usage, model structure, and reuse are closely tied to the studied DSLs. Similarly, some of the challenges reported by practitioners arise from the interface between BTs and ROS, rather than from BTs as a standalone model, and may manifest differently with other ecosystems. Finally, the two solutions we developed were evaluated with practitioners in session-based settings. As a result, we demonstrate their feasibility and perceived usefulness, but do not assess their impact on the overall quality of robotic missions or on development effort throughout a full cycle of robotic project development.

Beyond these boundaries, the broader applicability of our work lies in our analytical approach rather than in solely the quantitative results. We see parts of our built knowledge reaching further than BTs and transferring to other behavior-modeling languages. Investigating a behavior modeling language alongside its DSLs, linking language design decisions to practitioner usage, and reporting experiences and challenges are relevant to any behavior modeling

language in robotics. Our solutions are based on properties that BTs share with other behavior-modeling languages, such as raising the abstraction level, modularity and a transparent clear decision structure that facilitates stakeholder engagement. Where a behavior-modeling language provides these features and others, we anticipate that similar solutions can be implemented. In contrast, when a robot behavior coordination remains implicit within code, there is no shared model to convey quality or risk information. Finally, a broader lesson for robotics software engineering, the idea of using BTs and other behavior modeling languages in robotics as a boundary object and exploring new possibilities to support different stakeholders is an important insight of the thesis.

1.7 Conclusion and Future Work

The work in this thesis aimed to support the development of reliable robotic missions by drawing on software engineering practices. We started by building an understanding of existing support for specifying robotic missions using behavior modeling languages. We looked at concepts offered by different behavior modeling languages and, from that, established a research direction focused on BTs for the broader support they provide in terms of concepts and modeling openness compared to other behavior modeling languages. To ground our knowledge, we compared BT DSLs with SM DSLs and investigated their language engineering. After understanding the existing support in terms of offered concepts and the design of DSLs, we wanted to build knowledge about the adoption of BTs in practice. We started by investigating how the two behavior modeling languages in robotics, BTs and SMs, are adopted in open-source projects. Then, we conducted an empirical investigation into the experiences and practices of robotics practitioners when adopting BTs. Finally, by drawing on our built knowledge about BTs, we identified two potential gaps that BTs can address. We developed two solutions using BTs to support two key concerns in developing reliable robotic missions: quality and risk assessment. We provided multiple open-source repositories for BT and SM models, as well as our two developed solutions, to enable the research community to build on our results.

Future Directions: This thesis contributes to the development of reliable robotic missions by building comprehensive knowledge about the behavior modeling language, BTs, and creating solutions to address existing gaps. Throughout our discussion of the results, we highlighted potential research directions and recommendations for researchers and practitioners to improve the current state of practice. One future research direction is to construct a framework that integrates the various knowledge components and solutions presented in this thesis to support the full cycle of developing reliable robotic missions. At different stages of a robotic mission, various perspectives may be required depending on the stakeholders involved. We envision a framework that accommodates diverse stakeholders by representing their concerns using behavior modeling languages. We have already begun this journey by incorporating quality and risk considerations into the development cycle for robotic missions. By extending the use of our open-source resources, researchers can further develop the solution and support the explicit specification of other

important concerns. Izzo et al. [53] already took the first step by using our dataset of BT models from papers A and B to train lightweight large language models (LLMs) to generate BTs from natural-language mission specifications. We see this as an encouraging sign that the groundwork laid in this thesis can further open other directions of research.

Bibliography

[1] PwC, "The talent challenge: harnessing the power of human skills in the machine age," https://www.pwc.com/co/es/publicaciones/assets/ceo-survey-global-talent.pdf, 2017.

[2] T. Guardian, "Delivery robots are spreading across LA. residents 'both pity and hate them'," https://www.theguardian.com/us-news/2026/may/25/los-angeles-delivery-robots, 2026.

[3] J. Hawksworth, R. Berriman, and S. Goel, "Will robots really steal our jobs? an international analysis of the potential long term impact of automation," PricewaterhouseCoopers Australia, Tech. Rep., 2018.

[4] PwC, "Sizing the prize: What's the real value of AI for your business and how can you capitalise?" https://www.pwc.com.au/government/pwc-ai-analysis-sizing-the-prize-report.pdf, 2017.

[5] S. García, "Service robotics software engineering," Doctoral thesis, University of Gothenburg, Gothenburg, Sweden, 2021, department of Computer Science and Engineering, Division of Interaction Design and Software Engineering.

[6] S. Robotics, "Robotics 2020 multi-annual roadmap for robotics in europe," *SPARC Robotics, EU-Robotics AISBL, The Hauge, The Netherlands*, 2017.

[7] D. Brugali and E. Prassler, "Software engineering for robotics [from the guest editors]," *IEEE Robotics & Automation Magazine*, vol. 16, no. 1, pp. 9–15, 2009.

[8] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[9] S. Kolak, A. Afzal, C. Le Goues, M. Hilton, and C. S. Timperley, "It takes a village to build a robot: An empirical study of the ROS ecosystem," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 430–440.

[10] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[11] A. Nordmann, N. Hochgeschwender, and S. Wrede, "A survey on domain-specific languages in robotics," in *International conference on simulation, modeling, and programming for autonomous robots*. Springer, 2014, pp. 195–206.

[12] G. L. Casalaro, G. Cattivera, F. Ciccozzi, I. Malavolta, A. Wortmann, and P. Pelliccione, "Model-driven engineering for mobile robotic systems: a systematic mapping study," *Software and Systems Modeling*, vol. 21, no. 1, pp. 19–49, 2022.

[13] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[14] D. Faconti, "MOOD2Be: Models and tools to design robotic behaviors," European Union's Horizon 2020 Research and Innovation Programme, 2019. [Online]. Available: https://github.com/BehaviorTree/BehaviorTree.CPP/blob/master/MOOD2Be_final_report.pdf

[15] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis lectures on software engineering*, vol. 3, no. 1, pp. 1–207, 2017.

[16] "MDA is model-driven architecture technology," https://www.modelios oft.com/en/technologies/mda.html, 2022.

[17] S. Baerisch, "Use of models in software engineering," in *Domain-Specific Model-Driven Testing*. Springer, 2010, pp. 17–27.

[18] Object Management Group, "MDA guide revision 2.0," Object Management Group, Tech. Rep. ormsc/14-06-01, Jun. 2014. [Online]. Available: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01

[19] "UML 2.4 diagrams overview," https://www.uml-diagrams.org/uml-2 4-diagrams.html#structure-diagram, 2022.

[20] D. Brugali, "Model-driven software engineering in robotics: Models are designed to use the relevant things, thereby reducing the complexity and cost in the field of robotics," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 155–166, 2015.

[21] S. García, D. Strüber, D. Brugali, T. Berger, and P. Pelliccione, "Robotics software engineering: A perspective from the service robotics domain," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 593–604.

[22] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2208–2224, 2019.

[23] L. Bettini, *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.

[24] A. Wasowski and T. Berger, *Domain-specific Languages: Effective Modeling, Automation, and Reuse*. Springer, 2022. [Online]. Available: http://dsl.design

[25] N. Nilsson, "Teleo-reactive programs for agent control," *Journal of artificial intelligence research*, vol. 1, pp. 139–158, 1993.

[26] E. Gat, R. P. Bonnasso, R. Murphy *et al.*, "On three-layer architectures," *Artificial intelligence and mobile robots*, vol. 195, p. 210, 1998.

[27] M. Klauck, C. Henkel, M. Lampacrescia, and G. Jorgensen, "Surveying deliberation practices and methodological needs in robotics software engineering," in *Annual Conference Towards Autonomous Robotic Systems*. Springer, 2025, pp. 237–244.

[28] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artificial Intelligence*, vol. 247, pp. 10–44, 2017.

[29] J. F. Magee, *Decision trees for decision making*. Harvard Business Review Brighton, MA, USA, 1964.

[30] M. Colledanchise, "Behavior trees in robotics," Ph.D. dissertation, KTH Royal Institute of Technology, 2017.

[31] M. Iovino, J. Förster, P. Falco, J. J. Chung, R. Siegwart, and C. Smith, "Comparison between behavior trees and finite state machines," *IEEE Transactions on Automation Science and Engineering*, 2025.

[32] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto, "A survey of model driven engineering in robotics," *Journal of Computer Languages*, vol. 62, p. 101021, 2021.

[33] M. Radestock and S. Eisenbach, "Coordination in evolving systems," in *International Workshop on Trends in Distributed Systems*. Springer, 1996, pp. 162–176.

[34] K. Adam, A. Butting, R. Heim, O. Kautz, B. Rumpe, and A. Wortmann, "Model-driven separation of concerns for service robotics," in *Proceedings of the International Workshop on Domain-Specific Modeling*, 2016, pp. 22–27.

[35] C. Schlegel, A. Lotz, M. Lutz, and D. Stampfer, "Composition, separation of roles and model-driven approaches as enabler of a robotics software ecosystem," in *Software Engineering for Robotics*. Springer, 2021, pp. 53–108.

[36] A. Van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography," *ACM Sigplan Notices*, vol. 35, no. 6, pp. 26–36, 2000.

[37] M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. C. Kats, E. Visser, and G. Wachsmuth, *DSL engineering-designing, implementing and using domain-specific languages*. M Volter/DSLBook. org, 2013.

[38] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and AI," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022.

[39] I. Millington and J. Funge, *Artificial intelligence for games*. CRC Press, 2009.

[40] D. Isla, "Handling complexity in the Halo 2 AI," *GDC 2005 Proceedings*, 2005. [Online]. Available: https://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php?page=2

[41] R. G. Dromey, "From requirements to design: Formalizing the key steps," in *First International Conference on Software Engineering and Formal Methods, 2003. Proceedings.* IEEE, 2003, pp. 2–11.

[42] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard *et al.*, "An integrated system for autonomous robotics manipulation," in *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[43] F. Rovida, B. Großmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 6793–6800.

[44] A. Klöckner, "Interfacing behavior trees with the world using description logic," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2013.

[45] M. Hallen, M. Iovino, S. Sander-Tavallaey, and C. Smith, "Behavior trees in industrial applications: A case study in underground explosive charging," in *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE).* IEEE, 2024, pp. 156–162.

[46] G. Filippone, S. Pettinari, and P. Pelliccione, "Formalisms for robotic mission specification and execution: A comparative analysis," *IEEE*, 2026.

[47] P. Ögren and J. Alfredson, "Creating trustworthy AI for UAS using labeled backchained behavior trees," in *2023 International Conference on Unmanned Aircraft Systems (ICUAS).* IEEE, 2023, pp. 1029–1036.

[48] S. Dragule, E. Bainomugisha, P. Pelliccione, and T. Berger, "Effects of specifying robotic missions in behavior trees and state machines," *Journal of Computer Languages*, p. 101330, 2025.

[49] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[50] M. Colledanchise and L. Natale, "Analysis and exploitation of synchronized parallel executions in behavior trees," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2019, pp. 6399–6406.

[51] Q. Zhang, J. Yao, Q. Yin, and Y. Zha, "Learning behavior trees for autonomous agents with hybrid constraints evolution," *Applied Sciences*, vol. 8, no. 7, p. 1077, 2018.

[52] K.-J. Stol and B. Fitzgerald, "Guidelines for conducting software engineering research," in *Contemporary Empirical Methods in Software Engineering*. Springer, 2020, pp. 27–62.

[53] R. A. Izzo, G. Bardaro, and M. Matteucci, "Btgenbot: Behavior tree generation for robotic tasks with lightweight llms," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 9684–9690.

[54] M. AlMarzouq, A. AlZaidan, and J. AlDallal, "Mining GitHub for research and education: challenges and opportunities," *International Journal of Web Information Systems*, 2020.

[55] I. Malavolta, G. A. Lewis, B. Schmerl, P. Lago, and D. Garlan, "Mining guidelines for architecting robotics software," *Journal of Systems and Software*, vol. 178, p. 110969, 2021.

[56] G. Robles, T. Ho-Quang, R. Hebig, M. R. Chaudron, and M. A. Fernandez, "An extensive dataset of UML models in GitHub," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 519–522.

[57] S. García, D. Strüber, D. Brugali, A. Di Fava, P. Schillinger, P. Pelliccione, and T. Berger, "Variability modeling of service robots: Experiences and challenges," in *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems*, 2019, pp. 1–6.

[58] D. Brugali and A. Shakhimardanov, "Component-based robotic engineering (part ii)," *IEEE Robotics & Automation Magazine*, vol. 17, no. 1, pp. 100–112, 2010.

[59] I. A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I.-H. Shu *et al.*, "CLARAty: Challenges and steps toward reusable robotic software," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 5, 2006.

[60] D. Brugali and P. Scandurra, "Component-based robotic engineering (part i)," *IEEE Robotics & Automation Magazine*, vol. 16, no. 4, pp. 84–96, 2009.

[61] D. Kortenkamp, R. Simmons, and D. Brugali, "Robotic systems architectures and programming," in *Springer handbook of robotics*. Springer, 2016, pp. 283–306.

[62] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.

[63] V. B. Kampenes, B. Anda, and T. Dybå, "Flexibility in research designs in empirical software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. BCS Learning & Development, 2008.

[64] D. I. Sjøberg, T. Dybå, and M. Jørgensen, "The future of empirical methods in software engineering research." *FoSE*, vol. 7, no. 2007, pp. 358–378, 2007.

[65] J. S. Molléri and K. Petersen, "Teaching research design in software engineering," in *Handbook on Teaching Empirical Software Engineering*. Springer, 2024, pp. 71–100.

[66] R. Wieringa and A. Moralı, "Technical action research as a validation method in information systems design science," in *International Conference on Design Science Research in Information Systems*. Springer, 2012, pp. 220–238.

[67] B. Burger, P. M. Maffettone, V. V. Gusev, C. M. Aitchison, Y. Bai, X. Wang, X. Li, B. M. Alston, B. Li, R. Clowes *et al.*, "A mobile robotic chemist," *Nature*, vol. 583, no. 7815, pp. 237–241, 2020.

[68] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, 2004.

[69] F. Shull, J. Singer, and D. I. Sjøberg, *Guide to advanced empirical software engineering*. Springer, 2008, vol. 93.

[70] D. S. Cruzes and T. Dybå, "Recommended steps for thematic synthesis in software engineering," in *ESEM*. IEEE, 2011.

[71] J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," *IEEE software*, vol. 31, no. 3, pp. 79–85, 2013.

[72] O. M. Group, "OMG unified modeling language 2.5.1," 2017. [Online]. Available: https://www.omg.org/spec/UML/

[73] J. Bohren and S. Cousins, "The SMACH high-level executive [ROS news]," *IEEE Robotics & Automation Magazine*, vol. 17, no. 4, pp. 18–20, 2010.

[74] P. Schillinger, S. Kohlbrecher, and O. von Stryk, "Human-robot collaborative high-level control with an application to rescue robotics," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.

[75] S. García, D. Strüber, D. Brugali, A. Di Fava, P. Pelliccione, and T. Berger, "Software variability in service robotics," *Empirical Software Engineering*, vol. 28, no. 2, p. 24, 2023.

[76] N. Bencomo, J. Cabot, M. Chechik, B. H. Cheng, B. Combemale, S. Zschaler *et al.*, "Abstraction engineering," *arXiv preprint arXiv:2408.14074*, 2024.

[77] A. Bucchiarone, B. Combemale, A. Pierantonio, N. Bencomo, M. Van Den Brand, J.-M. Bruel, A. Cicchetti, J. Di Rocco, L. Lambers, J. Michael *et al.*, "Modeling: The heart and soul of engineering smart ecosystems," in *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2025, pp. 386–393.

[78] M. Fowler, *Domain-Specific Languages, Portable Documents.* Pearson Education, 2010.

[79] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Information and Software Technology*, vol. 59, pp. 67–85, 2015.

[80] M. P. Robillard and R. DeLine, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[81] T. Dybå, D. I. Sjøberg, and D. S. Cruzes, "What works for whom, where, when, and why? on the role of context in empirical software engineering," in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, 2012, pp. 19–28.

[82] B. A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-based software engineering," in *Proceedings. 26th International Conference on Software Engineering.* IEEE, 2004, pp. 273–281.

[83] A. Arab, A. Ferraro, A. Goodman, H. Asgari, J. Ibanez-Guzman, J. I. Olszewska, R. Calinescu, D. Scheidt, G. Mullins, and S. Redfield, "IEEE guide for verification of autonomous systems: An introduction to IEEE P2817," in *2026 IEEE Engineering Reliable Autonomous Systems (ERAS).* IEEE, 2026, pp. 167–173.

[84] J. Frattini, L. Montgomery, J. Fischbach, D. Mendez, D. Fucci, and M. Unterkalmsteiner, "Requirements quality research: a harmonized theory, evaluation, and roadmap," *Requirements engineering*, vol. 28, no. 4, pp. 507–520, 2023.

[85] IEEE Robotics and Automation Society, "IEEE P2817 Verification Of Autonomous Systems," https://www.ieee-ras.org/industry-activities/standards/active-projects/verification-of-autonomous-systems/, accessed 2026-07-07.

[86] D. Brugali and P. Salvaneschi, "Stable aspects in robot software development," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 4, 2006.

[87] J. M. Riley, L. D. Strater, S. L. Chappell, E. S. Connors, and M. R. Endsley, "Situation awareness in human-robot interaction: Challenges and user interface requirements," *Human-Robot Interactions in Future Military Operations*, pp. 171–192, 2010.

[88] V. Toncian, A. Florea, A. David, D. Morariu, and R. Cretulescu, "Leveraging collaboration for industry 5.0: Needs, strategies and future directions," in *Working Conference on Virtual Enterprises.* Springer, 2024.

[89] F. W. Heckel, G. M. Youngblood, and N. S. Ketkar, "Representational complexity of reactive agents," in *IEEE Conference on Computational Intelligence and Games (CIG).* IEEE, 2010.

[90] M. Colledanchise, A. Marzinotto, D. V. Dimarogonas, and P. Oegren, "The advantages of using behavior trees in multi robot systems," in *47th International Symposium on Robotics (ISR)*, 2016.

[91] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on robotics*, vol. 33, no. 2, pp. 372–389, 2016.

[92] M. Colledanchise, R. Parasuraman, and P. Ögren, "Learning of behavior trees for autonomous agents," *IEEE Transactions on Games*, vol. 11, no. 2, pp. 183–189, 2018.

[93] P. Ögren, "Increasing modularity of UAV control systems using computer game behavior trees," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2012.

[94] S. García, P. Pelliccione, C. Menghi, T. Berger, and T. Bures, "High-level mission specification for multiple robots," in *12th International Conference on Software Language Engineering (SLE)*, 2019.

[95] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[96] I. Bojic, T. Lipic, M. Kusek, and G. Jezic, "Extending the JADE agent behaviour model with JBehaviourTrees framework," in *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, 2011.

[97] N. Bencomo, R. B. France, B. H. C. Cheng, and U. Aßmann, Eds., *Models@run.time—Foundations, Applications, and Roadmaps*, vol. 8378. Springer, 2014.

[98] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *IEEE Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[99] "Paper A (SLE) and paper B (TSE) online appendix," https://bitbucket.org/easelab/behaviortrees, 2023.

[100] S. García, P. Pelliccione, C. Menghi, T. Berger, and T. Bures, "PROMISE: High-level mission specification for multiple robots," in *42nd International Conference on Software Engineering (ICSE), Demonstrations Track*, 2020.

[101] J. Chen and D. Shi, "Development and composition of robot architecture in dynamic environment," in *International Conference on Robotics, Control and Automation Engineering (RCAE)*, 2018.

[102] D. Stonier, N. Usmani, and M. Staniaszek, "Py trees library documentation," https://py-trees.readthedocs.io/en/devel/, 2020.

[103] D. Faconti and M. Colledanchise, "BehaviorTree.CPP library documentation," https://www.behaviortree.dev, 2018.

[104] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[105] D. Stonier, N. Usmani, and M. Staniaszek, "Py trees ROS library documentation," http://docs.ros.org/kinetic/api/py_trees_ros/html/index.html, 2020.

[106] M. Colledanchise, "BT++ library documentation," https://github.com/miccol/ROS-Behavior-Tree/blob/master/BTUserManual.pdf, 2017.

[107] E. Games, "Unreal engine 4 behavior tree library documentation," https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreeUserGuide/index.html, 2020.

[108] M. E. Conway, "Design of a separable transition-diagram compiler," *Commun. ACM*, vol. 6, no. 7, pp. 396–408, Jul. 1963.

[109] O.-J. Dahl and C. Hoare, "Hierarchical program structures," in *Structured Programming*, O.-J. Dahl, E. W. Dijkstra, and C. Hoare, Eds. Academic Press, 1972.

[110] E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen, "ABS: A core language for abstract behavioral specification," in *9th International Symposium on Formal Methods for Components and Objects (FMCO)*, 2010.

[111] J.-P. Tolvanen and S. Kelly, "How domain-specific modeling languages address variability in product line development: Investigation of 23 cases," in *23rd International Systems and Software Product Line Conference*, ser. SPLC, 2019.

[112] A. Alami, Y. Dittrich, and A. Wasowski, "Influencers of quality assurance in an open source community," in *11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2018.

[113] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," in *17th European Conference on Software Maintenance and Reengineering (CSMR)*, 2013.

[114] M. Colledanchise and P. Ögren, "How behavior trees modularize robustness and safety in hybrid systems," in *International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[115] F. Michaud and M. Nicolescu, "Behavior-based systems," in *Springer handbook of robotics*. Springer, 2016, pp. 307–328.

[116] K. Mcquillan, "A survey of behaviour trees and their applications for game AI," 2015, course CP5330 final report, James Cook University.

[117] O. Biggar, M. Zamani, and I. Shames, "An expressiveness hierarchy of behavior trees and related architectures," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5397–5404, 2021.

[118] M. Iovino, J. Förster, P. Falco, J. J. Chung, R. Siegwart, and C. Smith, "On the programming effort required to generate behavior trees and finite state machines for robotic applications," in *2023 IEEE International*

Conference on Robotics and Automation (ICRA). IEEE, 2023, pp. 5807–5813.

[119] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, and A. Wasowski, "Behavior trees in action: a study of robotics applications," in *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, 2020, pp. 196–209.

[120] S. Dragule, T. Berger, C. Menghi, and P. Pelliccione, "A survey on the design space of end-user-oriented languages for specifying robotic missions," *Software and Systems Modeling*, vol. 20, no. 4, pp. 1123–1158, 2021.

[121] M. Colledanchise and P. Ögren, "How behavior trees generalize the teleo-reactive paradigm and and-or-trees," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 424–429.

[122] D. C. Conner and J. Willis, "Flexible navigation: Finite state machine-based integrated navigation and control for ROS enabled robots," in *SoutheastCon 2017*. IEEE, 2017, pp. 1–8.

[123] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2718–2725.

[124] J. M. Zutell, D. C. Conner, and P. Schillinger, "Ros 2-based flexible behavior engine for flexible navigation," in *SoutheastCon 2022*. IEEE, 2022, pp. 674–681.

[125] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.

[126] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[127] CARLA Team, "ScenarioRunner for CARLA documentation," [Online]. Available: https://carla-scenariorunner.readthedocs.io, 2022, accessed: 2026-09-11. [Online]. Available: https://carla-scenariorunner.readthedocs.io

[128] M. L. Crane and J. Dingel, "UML vs. classical vs. rhapsody statecharts: not all models are created equal," *Software & Systems Modeling*, vol. 6, no. 4, pp. 415–435, 2007.

[129] M. Von der Beeck, "A comparison of statecharts variants," in *Formal techniques in real-time and fault-tolerant systems*. Citeseer, 1994, pp. 128–148.

[130] B. P. Douglass, "Chapter 5 - design patterns for state machines," in *Design Patterns for Embedded Systems in C*, B. P. Douglass, Ed. San Francisco, CA, USA: Newnes, 2011, pp. 257–356.

[131] M. Samek, *Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems.* CRC Press, 2008.

[132] H. Kubátová, K. Richta, and T. Richta, "Petri nets versus UML state machines," in *Proc. Conference on Software Development and Object Technologies (OBJEKTY)*, 2013, pp. 53–59.

[133] B. Hajji, A. Mellit, and L. Bouselham, "Finite state machines," in *A Practical Guide for Simulation and FPGA Implementation of Digital Design.* Springer, 2022, pp. 175–205.

[134] G. Lüttgen, M. Von der Beeck, and R. Cleaveland, "A compositional approach to statecharts semantics," *ACM SIGSOFT Software Engineering Notes*, vol. 25, no. 6, pp. 120–129, 2000.

[135] P. Schillinger, "An approach for runtime-modifiable behavior control of humanoid rescue robots," *Technische Universitat Darmstadt*, 2015.

[136] S. Kohlbrecher, A. Stumpf, A. Romay, P. Schillinger, O. von Stryk, and D. C. Conner, "A comprehensive software framework for complex locomotion and manipulation tasks applicable to different types of humanoid robots," *Frontiers in Robotics and AI*, vol. 3, p. 31, 2016.

[137] J. Bohren and S. Cousins, "SMACH library documentation," http://wiki.ros.org/smach/Documentation, 2010.

[138] P. Schillinger, "FlexBE library documentation," http://philserver.bplaced.net/fbe/documentation.php, 2016.

[139] D. Faconti and M. Colledanchise, "BehaviorTree.CPP library tutorials," https://www.behaviortree.dev/tutorial_01_first_tree/, 2018.

[140] D. Stonier, N. Usmani, and M. Staniaszek, "Py trees ROS library tutorials," https://py-trees-ros-tutorials.readthedocs.io/en/devel/tutorials.html, 2020.

[141] P. Schillinger, "FlexBE library tutorials," http://wiki.ros.org/flexbe/Tutorials, 2021.

[142] J. Bohren and S. Cousins, "SMACH library tutorials," http://wiki.ros.org/smach/Tutorials, 2021.

[143] M. Colledanchise and L. Natale, "On the implementation of behavior trees in robotics," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5929–5936, 2021.

[144] A. Romay, S. Kohlbrecher, A. Stumpf, O. von Stryk, S. Maniatopoulos, H. Kress-Gazit, P. Schillinger, and D. C. Conner, "Collaborative autonomy between high-level behaviors and human operators for remote manipulation tasks using different humanoid robots," *Journal of Field Robotics*, vol. 34, no. 2, pp. 333–358, 2017.

[145] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[146] T. Berger and J. Guo, "Towards system analysis with variability model metrics," in *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, 2014, pp. 1–8.

[147] J. A. Cruz-Lemus, M. Genero, and M. Piattini, "Using controlled experiments for validating UML statechart diagrams measures," in *Software Process and Product Measurement*. Springer, 2007, pp. 129–138.

[148] ——, "Metrics for UML statechart diagrams," in *Metrics for Software Conceptual Models*. World Scientific, 2005, pp. 237–272.

[149] R. van Tonder and C. Le Goues, "Lightweight multi-language syntax transformation with parser parser combinators," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 363–378.

[150] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *Proc. 13th International Conference on Embedded Software (EMSOFT)*. ACM, 2016, pp. 1–10.

[151] F. Rovida, M. Crosby, D. Holz, A. S. Polydoros, B. Großmann, R. Petrick, and V. Krüger, "SkiROS—a skill-based robot control platform on top of ROS," in *Robot operating system (ROS)*. Springer, 2017, pp. 121–160.

[152] F. Rovida and V. Krüger, "Design and development of a software architecture for autonomous mobile manipulators in industrial environments," in *2015 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2015, pp. 3288–3295.

[153] F. Rovida, "SkiROS2 library documentation," https://github.com/RVMI/skiros2/wiki, 2020.

[154] R. AI, "SMACC library documentation," https://smacc.dev/, 2018.

[155] M. Steinbrink, P. Koch, S. May, B. Jung, and M. Schmidpeter, "State machine for arbitrary robots for exploration and inspection tasks," in *Proceedings of the 2020 4th International Conference on Vision, Image and Signal Processing*, 2020, pp. 1–6.

[156] P. Laker, "Blackboard design pattern," https://social.technet.microsoft.com/wiki/contents/articles/13215.blackboard-design-pattern.aspx, 2012.

[157] E. F. Moore *et al.*, "Gedanken-experiments on sequential machines," *Automata studies*, vol. 34, pp. 129–153, 1956.

[158] J. Bosch, "Design patterns as language constructs," *Journal of Object Oriented Programming*, 1997.

[159] J. A. Cruz-Lemus, A. Maes, M. Genero, G. Poels, and M. Piattini, "The impact of structural complexity on the understandability of UML statechart diagrams," *Information Sciences*, vol. 180, no. 11, pp. 2209–2220, 2010.

[160] M. Genero, D. Miranda, and M. Piattini, "Defining and validating metrics for UML statechart diagrams," *Proceedings of QAOOSE*, vol. 2002, 2002.

[161] J. Cruz-Lemus, M. Genero, M. Piattini, and A. Toval, "Investigating the nesting level of composite states in UML statechart diagrams," *Proc. QAOOSE*, vol. 5, pp. 97–108, 2005.

[162] J. A. Cruz-Lemus, M. Genero, M. Piattini, and A. Toval, "An empirical study of the nesting level of composite states within UML statechart diagrams," in *International Conference on Conceptual Modeling*. Springer, 2005, pp. 12–22.

[163] L. Gherardi and D. Brugali, "Modeling and reusing robotic software architectures: The HyperFlex toolchain," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6414–6420.

[164] C. Schlegel, A. Lotz, M. Lutz, D. Stampfer, J. F. Inglés-Romero, and C. Vicente-Chicote, "Model-driven software systems engineering in robotics: covering the complete life-cycle of a robot," *it-Information Technology*, vol. 57, no. 2, pp. 85–98, 2015.

[165] D. L. Wigand, A. Nordmann, M. Goerlich, and S. Wrede, "Modularization of domain-specific languages for extensible component-based robotic systems," in *2017 First IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2017, pp. 164–171.

[166] C. Schlegel, A. Steck, D. Brugali, and A. Knoll, "Design abstraction and processes in robotics: From code-driven to model-driven engineering," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 324–335.

[167] C. Street, Y. Warsame, M. Mansouri, M. Klauck, C. Henkel, M. Lampacrescia, M. Palmas, R. Lange, E. Ghiorzi, A. Tacchella *et al.*, "Towards a verifiable toolchain for robotics," in *Proceedings of the AAAI Symposium Series*, vol. 4, no. 1, 2024, pp. 398–403.

[168] R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, and S. Dragule, "Behavior trees and state machines in robotics applications," *IEEE Transactions on Software Engineering*, vol. 49, no. 9, pp. 4243–4267, 2023.

[169] "Paper C (EMSE) online appendix," https://github.com/RazanGhzouli/Behavior-Trees-for-Robotic-Systems-data, 2026.

[170] F. Martín, J. G. Clavero, V. Matellán, and F. J. Rodríguez, "PlanSys2: A planning system framework for ROS 2," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 9742–9749.

[171] S. Bernagozzi, S. Faraci, E. Ghiorzi, K. Pedemonte, A. Ferrando, L. Natale, and A. Tacchella, "Model-based verification and monitoring for safe and responsive robots," in *2025 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE, 2025, pp. 1–6.

[172] E. Ghiorzi, C. Henkel, M. Palmas, M. Klauck, and A. Tacchella, "Execution semantics of behavior trees in robotic applications," *arXiv preprint arXiv:2408.00090*, 2024.

[173] V. DiLuoffo, W. R. Michalson, and B. Sunar, "Robot operating system 2: The need for a holistic security approach to robotic architectures," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, p. 1729881418770011, 2018.

[174] P. Estefo, J. Simmonds, R. Robbes, and J. Fabry, "The robot operating system: Package reuse and community dynamics," *Journal of Systems and Software*, vol. 151, pp. 226–242, 2019.

[175] J. W. Creswell, V. L. Plano Clark, M. L. Gutmann, and W. E. Hanson, "Advanced mixed methods research designs," *Handbook of mixed methods in social and behavioral research*, vol. 209, no. 240, pp. 209–240, 2003.

[176] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[177] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *2009 3rd international symposium on empirical software engineering and measurement*. IEEE, 2009, pp. 401–404.

[178] K. Säfsten and M. Gustavsson, *Research methodology: for engineers and other problem-solvers*. Studentlitteratur AB, 2020.

[179] M. Dahl, E. Erős, K. Bengtsson, M. Fabian, and P. Falkman, "Sequence planner: A framework for control of intelligent automation systems," *Applied Sciences*, vol. 12, no. 11, p. 5433, 2022.

[180] D. F. Vears and L. Gillam, "Inductive content analysis: A guide for beginning qualitative researchers," *Focus on Health Professional Education: A Multi-Professional Journal*, vol. 23, no. 1, pp. 111–127, 2022.

[181] E. Erős, K. Bengtsson, M. Dahl, and P. Falkman, "A framework for integrated virtual preparation and commissioning of intelligent automation systems," *Available at SSRN 4157292*, 2022.

[182] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "The (r) evolution of social media in software engineering," *Future of software engineering proceedings*, pp. 100–116, 2014.

[183] A. Begel, R. DeLine, and T. Zimmermann, "Social media for software engineering," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010, pp. 33–38.

[184] K.-J. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, pp. 1–51, 2018.

[185] G. Terry, N. Hayfield, V. Clarke, V. Braun *et al.*, "Thematic analysis," *The SAGE handbook of qualitative research in psychology*, vol. 2, no. 17-37, p. 25, 2017.

[186] M. TENORTH, "Controlling process of robots having a behavior tree architecture," May 24 2022, uS Patent 11,338,434.

[187] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior trees," in *2019 international conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 8839–8845.

[188] F. M. Rico, M. Morelli, H. Espinoza, F. J. Rodríguez-Lera, and V. M. Olivera, "Optimized execution of PDDL plans using behavior trees." in *AAMAS*, 2021, pp. 1596–1598.

[189] J. F. Maier, C. M. Eckert, and P. J. Clarkson, "Model granularity in engineering design–concepts and framework," *Design Science*, vol. 3, p. e1, 2017.

[190] D. L. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.

[191] I. Malavolta, G. Lewis, B. Schmerl, P. Lago, and D. Garlan, "How do you architect your robots? state of the practice and guidelines for ROS-based systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, 2020, pp. 31–40.

[192] P. Canelas, B. Schmerl, A. Fonseca, and C. S. Timperley, "Understanding misconfigurations in ROS: an empirical study and current approaches," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 1161–1173.

[193] J. Ao, F. Wu, Y. Wu, A. Swiki, and S. Haddadin, "LLM-as-BT-planner: Leveraging LLMs for behavior tree generation in robot task planning," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 1233–1239.

[194] H. Zhou, Y. Lin, L. Yan, J. Zhu, and H. Min, "Llm-bt: Performing robotic adaptive tasks based on large language models and behavior trees," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 16 655–16 661.

[195] E. Dortmans and T. Punter, "Behavior trees for smart robots practical guidelines for robot software development," *Journal of Robotics*, vol. 2022, no. 1, p. 3314084, 2022.

[196] E. S. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *3rd IEEE Int'l Symp. on Requirements Eng.*, 1997, pp. 226–235.

[197] M. Vierhauser, R. Rabiser, and P. Grünbacher, "Requirements monitoring frameworks: A systematic review," *Information and Software Technology*, vol. 80, pp. 89–109, 2016.

[198] "Paper D (REFSQ) online appendix," https://github.com/RazanGhzouli/qualityBTDSL, 2024.

[199] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer Science & Business Media, 1999, vol. 5.

[200] J. Eckhardt, A. Vogelsang, and D. Méndez Fernández, "Are "non-functional" requirements really non-functional? an investigation of non-functional requirements in practice," in *38th Int'l Conf. on Software Eng.*, 2016, pp. 832–842.

[201] International Organization for Standardization, "ISO/IEC 25010:2023 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model," https://www.iso.org/standard/78176.html, 2023.

[202] H. A. Simon, "Rational choice and the structure of the environment." *Psychological review*, vol. 63, no. 2, p. 129, 1956.

[203] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[204] S. Efftinge and M. Völter, "oAW xText: A framework for textual DSLs," in *Workshop on Modeling Symp. at Eclipse Summit*, vol. 32, no. 118, 2006.

[205] G. Guest, K. M. MacQueen, and E. E. Namey, *Applied thematic analysis*. sage publications, 2011.

[206] I. Dejanović, R. Vaderna, G. Milosavljević, and Ž. Vuković, "Textx: a python tool for domain-specific languages implementation," *Knowledge-based systems*, vol. 115, pp. 1–4, 2017.

[207] A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, "Grand challenges in model-driven engineering: an analysis of the state of the research," *Software and Systems Modeling*, vol. 19, pp. 5–13, 2020.

[208] E. Giunchiglia, M. Colledanchise, L. Natale, and A. Tacchella, "Conditional behavior trees: Definition, executability, and applications," in *2019 IEEE Int'l Conf. on Systems, Man and Cybernetics (SMC)*, 2019, pp. 1899–1906.

[209] G. De Campos Affonso, K. Okada, and M. Inaba, "State-aware layered BTs—behavior tree extensions for post-actions, preferences and local priorities in robotic applications," in *Int'l Conf. on Intelligent Autonomous Systems*, 2022, pp. 673–689.

[210] A. Steck and C. Schlegel, "Towards quality of service and resource aware robotic systems through model-driven software development," *arXiv preprint arXiv:1009.4877*, 2010.

[211] M. Reichardt, T. Föhst, and K. Berns, "On software quality-motivated design of a real-time framework for complex robot control systems," *Electronic Communications of the EASST*, vol. 60, 2013.

[212] V. Monthe, L. Nana, G. E. Kouamou, and C. Tangha, "RsaML: A domain specific modeling language for describing robotic software architectures with integration of real time properties." in *EWiLi*, 2016.

[213] J. Guiochet, M. Machin, and H. Waeselynck, "Safety-critical advanced robots: A survey," *Robotics and Autonomous Systems*, vol. 94, 2017.

[214] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, vol. 86, no. 6, 2013.

[215] H. Foidl, M. Felderer, and S. Biffl, "Technical debt in data-intensive software systems," in *SEAA*. IEEE, 2019.

[216] P. Curtis, M. Carey, C. of Sponsoring Organizations of the Treadway Commission *et al.*, "Risk assessment in practice," American Institute of Certified Public Accountants, Tech. Rep., 2012.

[217] "Paper E (ETFA) online appendix," https://github.com/RazanGhzouli/2025-BT-risk-assessment/, 2025.

[218] *Directive 2006/42/EC of the European Parliament and of the Council on Machinery, and Amending Directive 95/16/EC (recast)*, European Parliament and Council of the European Union, Brussels, May 2006, official Journal of the European Union.

[219] *Safety of machinery – General principles for design – Risk assessment and risk reduction*, International Organization for Standardization, Geneva, Switzerland, November 2010.

[220] M. Bdiwi, I. Al Naser, J. Halim, S. Bauer, P. Eichler, and S. Ihlenfeldt, "Towards safety4. 0: A novel approach for flexible human-robot-interaction based on safety-related dynamic finite-state machine with multilayer operation modes," *Frontiers in Robotics and AI*, vol. 9, 2022.

[221] N. Banduka, I. Veza, and B. Bilić, "An integrated lean approach to process failure mode and effect analysis PFMEA: A case study from automotive industry," *Advances in Production Engineering & Management*, vol. 11, no. 4, 2016.

[222] H. Schneider, "Failure mode and effect analysis: FMEA from theory to execution," 1996.

[223] D. Battini, M. Calzavara, A. Persona, and F. Sgarbossa, "A comparative analysis of different paperless picking systems," *Industrial Management & Data Systems*, vol. 115, no. 3, 2015.

[224] A. Abdulkhaleq and S. Wagner, "Integrating state machine analysis with system-theoretic process analysis," in *Software Engineering 2013-Workshopband*. Gesellschaft für Informatik eV, 2013.

[225] G. Kokotinis, G. Michalos, Z. Arkouli, and S. Makris, "A behavior trees-based architecture towards operation planning in hybrid manufacturing," *International Journal of Computer Integrated Manufacturing*, vol. 37, no. 3, 2024.

[226] L. Castano and H. Xu, "Safe decision making for risk mitigation of UAS," in *ICUAS*. IEEE, 2019.

[227] S. G. Hart, "NASA-task load index NASA-TLX; 20 years later," in *Proc. of the human factors and ergonomics society annual meeting*, vol. 50, no. 9. Sage publications Sage CA: Los Angeles, CA, 2006.

[228] Otter.ai, "Otter.ai: Transcription service," accessed April 23, 2025.

[229] J. Hutchinson, J. Whittle, and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," *Science of Computer Programming*, vol. 89, 2014.

[230] R. Wohlrab, E. Knauss, J.-P. Steghöfer, S. Maro, A. Anjorin, and P. Pelliccione, "Collaborative traceability management: a multiple case study from the perspectives of organization, process, and culture," *Requirements Engineering*, vol. 25, no. 1, 2020.

[231] H.-C. Liu, L. Liu, and N. Liu, "Risk evaluation approaches in failure mode and effects analysis: A literature review," *Expert systems with applications*, vol. 40, no. 2, 2013.