ISSN (e): 2250 – 3005 || Volume, 15 ||Issue, 11||November-2025 ||
International Journal of Computational Engineering Research (IJCER)

Analytical Evaluation of Research Methods in Computer Science Theses

Richard Omollo
Department of Computer Science and Software Engineering
Jaramogi Oginga Odinga University of Science and Technology, KENYA

ABSTRACT
This research paper critically examines effective research methodologies applicable to Computer Science (CS) theses, emphasizing methodological rigor, epistemological awareness, and the integration of empirical, theoretical, and formal paradigms. It argues that many CS theses lack sufficient methodological grounding, often blending engineering design with scientific inquiry. Leveraging from a researcher's experiential perspective, the paper describes how design, experimentation, and theoretical formulation converge to produce validated knowledge contributions. By adopting a social constructivist and pragmatic experimental stance, it underscores the importance of aligning research design with philosophical commitments, ensuring reliability, and reinforcing validity through appropriate measurement and statistical inference. The paper further evaluates the literature review as an independent research instrument, identifying its role in synthesizing prior work to frame novel contributions. It also highlights the interplay between prototype development, reproducibility, and open science practices, advocating for ethical awareness and transparent documentation. Ultimately, the discussion provides a coherent framework for structuring and executing CS theses that balance creativity with scientific discipline, fostering methodological clarity and epistemic contribution. The insights presented are intended to guide postgraduate researchers in developing methodologically sound, empirically grounded, and theoretically robust CS theses that advance both academic scholarship and practical innovation.
KEYWORDS: *Computer Science Research Methods, Methodological Rigor, Empirical Software Engineering, Design Science Research, Reproducibility, Knowledge Contribution*

Date of Submission: 03-11-2025 Date of acceptance: 13-11-2025

I. Introduction

Thesis and dissertation work often exhibits insufficient methodological rigor, undermining the credibility and impact of research claims. Computer Science (CS) is particularly affected due to the tendency to regard engineering design as an end in itself. Early reflections on thesis writing also reveal views on this topic that are, at best, incomplete, providing poor guidance for framing questions and claims in CS. Establishing an understanding of effective, specific research methodology necessitates a perspective that sees design as a means for producing knowledge rather than as a fundamental goal. The term "knowledge" is approached in a broad sense, covering concept formulation, experimental results, or the establishment of construction teachings. The search for a guiding lens leads to the pronouncements of: "knowledge contribution" and "the practice of" from a prominent researcher [1] ; and the identification of a "knowledge culminators" perspective by a prominent educator, which clarifies that a research undertaking "culminates in a knowledge statement."

1.1. Rationale for Methodological Rigor in CS Theses

Master's and doctoral theses play a pivotal role in academia: they reflect a researcher's most important contributions and unveil the transition from apprentices to independent scholars. Despite the prevalence of computer science (CS) theses, only fragmentary guidance on their preparation exists. Bad ideas are often propagated unchecked. Systems scarcely considered by others merit extensive methodological development. The four preliminary sections articulate the challenges confronted and the strategies adopted, viewed from the researcher's yet personal standpoint—intended, however, not as solipsism or self-aggrandizement, but merely an anchoring frame for orienting the questions raised and the claims ventured.

The specific choice of terminology in qualifying a thesis as CS hinges on conventions within the field. Computer science, like many disciplines, encompasses a plurality of subfields in which global research problems and dominant paradigms drift apart. When interesting ideas unfold, the delineation may unintentionally impose restrictions inappropriate for a specific context, driving the researcher toward ill-conceived alternatives or hindering progress altogether. Subsequent sections furnish a general exposition still governed, due to the intertwined nature of the domains, by CS standards firmly rooted within empirical software engineering, diverse theoretical and formal approaches, and the application of varied analytical evaluation metrics.

The accompanying reflections thus concerns challenges encountered and strategies pursued—articulated through an experiential lens. The set of issues addressed resonates beyond the immediate scope; other embedded questions persist under consideration, yet the narrative admits no straightforward way of orchestrating them in an alternative order. [2] [3]

1.2. Positioning the Researcher's Perspective

A thesis statement, while crucial, is not exclusively responsible for defining the study's contribution, nor does its absence preclude identifying a question. Some researcher's approaches sufficiently narrow the effort to afford claim formulation. Insight often arises from interrogation, but framing that remains tentative leaves an exploration open, enabling an audience from adjacent disciplines or other fields to appreciate uncertainty. Even for problems generally acknowledged as important, primacy sometimes wishes to avoid subjective prioritisation. Rather than insisting on the determinant question from the start, it can prove fruitful to specify, through a few candidates, what space of inquiry the interlaced investigations occupy. A framing query that a reader unfamiliar with the area can appreciate adds value. Prior authors refer to this practice as "framing the research question" or the "problem statement''; "guiding questions" signals a difference by conveying openness to adjustment and the tentative nature of the existing formulation [1].

In computer science (CS), the nature of the question influences the methods employed and the anticipation of both input and output that guides the undertaking. Three broad modes of research prevail: empirical, theoretical, and formal [4]. Each sits on a space formed by dimensions such as development of (empirical) implementations or (theoretical) notational systems, exposition at abstract (formal) algorithmic level or in terms of (theoretical) data and control structures, inclination toward simulation or simple, coarsely calculable models, and (theoretical) adherence to independence of implementation. Depending on topic, it is possible to pass to a more or less restricted subdivision of established paradigms or genres. Commonly appearing aspects or especially relevant aspects of the technique complement the overall position statement and inform audience expectations. Emphasizing particular facets of combination and the atypical blending merits attention in the presentation as well.

II. Research Design and Philosophical Foundations

Many researchers in the fields of computing and computer science adopt the epistemological position of social constructivism. In simple terms, social constructivism is a theory that posits that a person's understanding of the world is built through their socio-cultural experiences, as opposed to being a universally valid "truth". A researcher who adopts the social constructivist position, rather than believing it is possible to remain entirely objective and neutral, is instead aware of their background and how it shapes their perspective. The researcher's viewpoint, at the point of writing, is an individual who possesses a deep understanding of information systems and extensive experience with software engineering.

In the framework of the 20th century philosopher Thomas Kuhn, a discipline such as computer science may be characterized as a normal science rather than a revolutionary paradigm. Most CS research is performed when social problems emerge that technology fails to solve. The knowledge required to develop effective solutions is often drawn from multiple fields; tackling recognition of handwritten Arabic, for example, may involve legal, political, linguistic, and socio-economic knowledge. Computer scientists then work within the framework of existing technology to address the problem. Formal theories, however, describe a fully-developed solution that meets all the specifications. Formalism tends to separate computer science formalism from that of mathematics and distinguishes it from other design-based fields.

Within the paradigm of social constructivism, a researcher benefits from being able to connect with explained knowledge and solutions within their field of expertise. As the discipline of computer science arises from the intersection of many fields rather than being entirely supplementary, computer scientists commonly contribute not only to novel technology but also to existing epistemic knowledge. Counter-knowledge concerning frameworks, instruments, and other commonplace abstractions exists freely on the web. The standpoint of formalism, entity life-cycle management, and other parameters of database-management system theory are widely shared. Consequently, as leaders anticipate yet another major paradigm shift, fundamental simplifications may be achievable during the transition. [5]

2.1. Epistemological Stance and Its Implications

Research demands rigorous methodological attention even in creative domains. The following sections frame the discussion through the lens of a research perspective fostered by a formal CS education. To address a problem, articulate a milieu and examine prior work at the intersection of both aspects. Research design emerges from these considerations, encompassing a literature review as a pivotal instrument of inquiry. Validation of novel propositions relies on careful identification of fitting experimental, theoretical, or formal methods appropriate to the domain.

Mature disciplines expect scholars to position arguments within agreed portfolios of accepted knowledge, conditioned by accepted definition of knowledge and epistemological commitment. Crafting a CS thesis necessitates grappling with epistemological matters, engaging with a particular disputation about the field in its infancy [6]. Such clarifications underpin assurance of rigorous achievement, striking an appropriate balance between valid contributions to established knowledge and overall novelty. Prior exposure also enhances sensitivity to literary style, structure, and complementary materials favouring adoption of a more relaxed, reflexive presentation [7].

2.2. Choosing a Research Paradigm

Research methodology in computer science (CS) encompasses a broad spectrum of paradigms, instruments, strategies, and techniques. A researcher requires a coherent, convincing narrative linking these choices and aligning them with established practices in the CS community. Such transparency strengthens the position of the research study, clarifies its assertions, and enhances its pedigree.

At the philosophical level, the study adopts an agnostic empiricist epistemology. The chosen research paradigm is considered a pragmatic experimental approach. This framework is defined as “building, designing, creating, generating, making, or inventing a solution to a problem, followed by observing or measuring the resulting behavior” [8]. The CS-specific activities of designing, specifying, and implementing a software prototype are framed as experimental investigations. The agnostic aspect emphasizes the independence of any philosophical commitment to science, gnosiology, or the role of preliminary considerations in the pursuit of scientific knowledge. The nature of this knowledge, the obstacles preventing its acquisition, and the corresponding degree of certainty remain open questions.

Some researchers encounter difficulty connecting experimental inquiries to the central question of how knowledge constitutes a secured scientific contribution. The study interrogates literature reviews as a separate general-purpose research instrument. It thereby clarifies the nature of the study itself and the eventual knowledge it is prepared to deliver.

2.3. Study Validity and Reliability in CS

In empirical software engineering, as in other experimental disciplines, studies face validity threats that all scholars should anticipate and mitigate [9]. The concept of validity itself encompasses various interpretations, but studies generally aim at establishing a relationship, illuminating a causal chain, or disclosing correlations in data sets. Thus, valid studies measure phenomena closely related to the research claims; employing student samples to investigate practices employed exclusively by professionals, for instance, would introduce serious concerns. Remaining within the domain of computer science, this section identifies candidate metrics, potential measurements, suitable statistical inference plans, and widely recognized threats, together with strategies for alleviating them within the confines of the discipline [10].

Although teaching, pedagogy, or education simply directs empirical attention toward a specific population, it may also profoundly shift the very nature of the inquiry, assuming educational concepts specifically appear in the central research questions. Therefore, here, systematic and exhaustive enumerations of metrics and respective measurements accompanying formal identification of a theoretical problem reiterate the topics already covered.

III. Literature Review as a Research Instrument

Acquainting oneself with prior work by systematically surveying the literature constitutes an instrument of research unto itself. Particularly in a rapidly evolving field like software engineering, such a survey seeks not only comprehensiveness but also distinctiveness: although the constituent elements and the established framework may remain within well-known bounds, synthesizing those preexisting components in an innovative way—whether within a new application domain, by applying a different real-world perspective, or via a new organization of the material—requires precisely the same analytic capacity needed to grasp the collective state of knowledge and identify concrete proposals for improvement, for which a doctoral candidate is already expected to demonstrate adequate preparation [11]).

The strategies pursued in preparing the current work for defense illustrate surveying on this scale. Even limiting consideration to papers with an explicitly computer-science dimension, the number of potentially pertinent works spanned over 350 published items. Addressing the excluding criterion of relevance, the comparable total drops to roughly one hundred, including software-distribution schemes, development methodologies, project-management approaches, model-driven techniques, process frameworks, and process-standard implementations. Excluded items are those lacking a direct connection, whose software-engineering link resides solely within the application domain, or whose proposal pertains primarily to delivering expertise rather than acquiring it.

Illustrating prior contributions within the current document calls for retaining full data on both cited and uncited results throughout the composition, lest the inevitable onset of familiarity with quoted materials impair the capacity to evaluate their value in summary form. Reliable automation at this level eludes present tools, and manual upkeep amounts to impractical overhead.

Handling such content independently, either in explanation or in the equal treatment implicit in subsection citations, undeniably complicates clear positioning of a proposed proposition vis-à-vis existing knowledge. To unify the exposition around the title allegation without sacrificing the necessary sociotechnical distinctions, prior knowledge remains at the forefront in ordering both the ensuing sections and their enclosed component paragraphs.

3.1. Strategies for Comprehensive Surveying

Effective research methods tend to follow established guidelines or the methods of established researchers. The selected methods, however, should be individualised according to the particular study. The researcher's perspective has concluded that the field of Computer Science is not solely about programming. Programmers tend to collect evidence, write code, and document to illustrate what has been done. In contrast, the selected methods focus on collecting and analysing evidence primarily outside of the programming domain. Within Computer Science, a literature review can be an effective instrument for assembling evidence and documenting that evidence and its analysis remains critically important in a Computer Science study. The study of previous work helps to derive future work, whether it be creating a synthesis with a novel idea, or simply gathering papers that are related or even contradictory to the own work. Applied Computer Science exhibits a variety of software engineering studies that have explored those options. The conclusion is that an adequate and effective literature review is both research and empirical work that augments the title of Computer Science with empirical obligations and remains profound enough to free the study from relying solely on surface examination.

As noted, the literature review can assume many forms including state-of-the-art surveys, state-of-the-practice surveys, systematic literature reviews, systematic mapping studies and the application of material science data on modelling and measuring software in general. A literature review may therefore be a longitudinal survey. For the first reporting of the review, a longitudinal state of the art survey has been conducted. The literature review is ongoing as new material continually emerges although it has been formalised and limited for a second reporting. Software Engineering follows its own path within Computer Science and remains a fundamental area that quarantines literature reviews in sufficient detail. The analyses were conducted for the identification of those themes. Computer Science experimentation can comprise several different branches among them software engineering, security, verification and validation and scripting. The number of documentation themes prevails in literature concentrates on the first two yet many Computer Science degrees encompass software engineering modelling, analysis software engineering and similar practical fields. The following challenges aid in maintaining the Computer Science perspective.

3.2. Synthesizing Prior Work with Novelty

Despite always being present, novelty does not play an equally important role in every thesis. An incremental improvement over existing methods can significantly impact areas with practical difficulties, such as software engineering. Novelty motivates students to select relevant thesis topics and assists the audience in understanding the underlying thought processes. A literature review reveals gaps in existing formulations, providing a solid starting point for the elaboration phase. Addressing previous works within the main thesis text—bypassing the requirements of a separate state-of-the-art section—is beneficial, often allowing for revisions of clearer choices or overall cate-gorisation. Reinforcing these questions has sharpened the formulation from the very beginning and yielded further ideas about the design of the carried evaluation. Several complementary observations presented here will also support the exposition of software-engineering–related thesis work, where experiments deploy a prototype, log data, come with a subsequent analysis, and rely on benchmarking sets and comprehensive state-of-the-art documents [1]. The message aligns with the gap pursued yet implies a different conceptualisation. Novelty, restricted to scientific aspects, cannot be absolute for all these propositions. The central aim remains to provide a contribution; framing the study as purely incremental would subsequently reduce its visibility. So-called radical and incremental novelties exist. The preceding literature work illustrates already a gap aligned with the central study, which thus still advances knowledge without introducing altered interfaces or languages. Further clarification regarding the connectedness of similar yet different design decisions across diverse systems completes the picture. Amid software-engineering approaches, algorithms invariably rank in a secondary role, although machine-learning publications practically bring the topic back to centre stage—compensating perhaps the earlier omission stemming from topic choice [12].

IV. Empirical Methods in Computer Science

Finding the right empirical research methods is essential for effective thesis work. In computer science, empirical methods remain relatively unexplored. Software engineering exemplifies need for guidance on experimental design and measurement in addition to the analysis of collected data. Empirical studies utilize replication, controlled and natural experimentation, quasi-experimentation, surveys, case studies, observational studies, ethnography, simulation, and models [13]. Thesis work may include experimentation but rarely supports software-engineering practices because corresponding methods are poorly documented across the field.

Thesis research typically generates prototype software as a contribution. In software engineering, prototype design, construction, and experimentation constitute an exploratory form for formulating questions, models, and tools and testing hypotheses [1]. Software-engineering literature still lacks sufficient detail for tracking and justifying prototypes' development, although theories about models, questions, and experiments provide some insight into the rationale that underlies and connects these activities.

4.1. Experimental Design for Software Engineering

Experimental design encompasses a rigorous process for evaluating a new software engineering technique and determining its effectiveness compared with existing approaches. The investigation typically begins with a theoretical analysis in the form of a literature review or a presentation of relevant algorithmic schema [1]. Building on these foundations, one or more experimental studies are conducted to collect empirical evidence. For such studies, the researcher must define a benchmarking technique and a data set, ensuring that sufficient information is provided to allow others to replicate the study independently. These aspects are crucial for sustaining progress in the field by enabling knowledge sharing and independent verification of claims [14].

Experimental studies commonly involve a detailed prototype of the new technique that is deployed on a selection of data sets. The experimental design and the relevance of the data sets are then presented to substantiate a claim regarding the theoretical progression achieved [15]. Prototype development necessitates specifying input and output formats for the data sets, as well as establishing the hardware and software environments employed by the prototype. Explicitly detailing these factors facilitates understanding of the technique and underscores its significance.

4.2. Benchmarking, Data Sets, and Reproducibility

In software engineering, a design experiment assesses the impact of a systematic manipulation of specified factors on an observed response [16]. The manipulations can occur on a single prototype or through a full factorial comparison of two or more contrasting prototypes. Data sets and benchmarking are closely linked to the establishment of experimental treatments and observations, and thus have a strong influence on the overall experimental design. Benchmarking establishes knowledge about how a design choice or decision affects resource-utilization characteristics, and data sets provide extant data whose characteristics stimulate specific designs. Since the definitions of data sets and benchmarking are coupled with reproducibility, both topics are addressed in conjunction with prototype development; the third aspect of experimental design.

Benchmarking is defined as "a set of based measurements used for comparisons" [17]. Hence, a datum, itself a measurement, is not a benchmark, whereas a set of measurements on a given topic constitutes benchmark. With respect to data sets, a benchmark comprises a collection of a variety of data sets that share an identifiable feature or property, with accompanying data. An experimental design is reproducible if successive occurrences generate identical observations when exposed to the same conditions, and involves both a physical and logical copy of the system under study in compliance with established protocols.

4.3. Prototype Development and Evaluation

Design science research (DSR) is a constructive research approach that combines knowledge-related and design-related questions. Despite DSR's appropriateness for the combination of theoretical and prototype development, it is seldom employed in computer science master's theses across software engineering and human-computer interaction. The limitations of standard practice may arise from a lack of examples, directive guidance, and author experience. Students commonly believe solid theoretical foundations are required prior to prototype implementation; however, prototyping may lead to new insights and shifted questions that enhance knowledge acquisition. The methodological alignment between knowledge gain and prototype development proved beneficial in earlier doctoral work [1].

Prototypes usually serve to explore the suitability of a concept or idea and implement the respective functionalities with less effort than a fully-featured, industrial-grade realization or similar high-quality research artefacts. When synthesis of theoretical contributions and practical artefacts proves challenging, an interim solution encompasses design and implementation of small and precise prototypes without obtaining deep understanding in advance. Development proceeds gradually based on existing knowledge and concepts; pilots involve only minor theoretical effort to address modelling aspects, thus accelerating both development and comprehension compared to purely theoretical methods. Once the implementation reaches a certain level of completeness, a more comprehensive understanding emerges that allows questions to be framed at a higher abstraction level, yielding new insights or material for further exploration.

Evaluation serves to test how well the implementation adheres to the original design, examining additional characteristics beyond correctness. Guided by the core questions of the thesis, prototypes not only facilitate knowledge acquisition but also actively trigger further questions that merit investigation. Borrowing from DSR, prototype development and evaluation constitute important building blocks within the overall research process and contribute to relevant knowledge gain. When applying DSR, design requires comprehensive modelling efforts in disciplines such as simulation, computer security, and compilers. Within information systems, computer science, educational aids, and similar fields, it is possible to develop and evaluate development-oriented prototypes without extensive modelling groundwork.

Prototype demonstrations conveniently address formal modelling concepts and facilitate teaching, while executable code fosters direct interaction with the model, enabling users to explore behaviour without specifying precise input or parameters. This aspect clarifies interdynamics among components within formal model development. Knauss found initial modelling rules beneficial yet restrictive; employing modelling merely to validate correctness precluded open-ended exploration of diverse properties. At the doctoral level, Knauss opted for a modelling-free demonstration oriented towards overall dynamics instead. Programming-language knowledge and design patterns governing formal models facilitate comprehension of the undefinability–decidability–completeness interplay, enabling straightforward illustration of the paradigm without extensive introductory effort.

V. Theoretical and Formal Methods

Formal and semi-formal methods study abstract models of computational systems through logical reasoning and mathematical proofs. In theoretical computer science, algorithmic analysis and proof of correctness establish fundamental properties of algorithms and programs [18]. Complexity theory characterizes the computational cost of solving decision problems and defines relationships between problems; nontrivial hardness results support the significance of chosen problems in a thesis. A literature review should survey all complexity aspects relevant to the problems addressed.

Rigorous methods can demonstrate crucial properties of a program or system with absolute certainty [1]. Prototypes, often the only available implementations, require appealing evaluation or supplementary proofs. Evaluation remains a valid concern when scope or effort prevent full implementation, and notions of completeness differ between analysis and implementation.

5.1. Algorithmic Analysis and Proof Techniques

Algorithmic analysis, design and analysis of algorithms occupy a central position in computer science. The standard framework for performance analysis of algorithms is to define an appropriate idealized machine model on which the algorithm will be executed and to examine the resource usage of the algorithm with respect to that model. Resource requirements usually considered include execution time (or steps) and space (or storage units) needed to run an algorithm on an input of size n. The running time of an algorithm with respect to the the adopted machine model must correlate well with real-life execution time of the algorithm hence it is also indispensable to precisely describe other characteristics of the machine model adopted and the required input in order to make meaningful performance comparisons. Many complexities measures can be defined over different structures yielding many acceptable uniformity frameworks yet as an example both vertex and edge set can be considered inputs for algorithms seeking to construct a spanning tree. Any particular algorithm belongs to a complexity class defined by all instances where it produces a correct systematic output. The communicating complexity of a protocol is a hundredth of a second in communication rounds. The class structurally describes those algorithms needing similar amounts of interaction. A characterization of such extensive properties of method defines its nondeterministic time bound relative to a Turing machine with n-tape and k states. For self-contained presentation, Turing machines, complexity measure and interaction and space need to be introduced even if the research does not concern these subjects explicitly [2].

5.2. Complexity Considerations and Hardness Results

Computer science investigations often aim to establish fundamental claims [19]. Such assertions about specific methods, representations, or protocols typically involve a combination of two considerations: the relative complexity involved in executing these methodologies, and the extent to which problems remain intractable within the designated framework [20]. Complexity theory offers a variety of precise techniques for establishing the difficulty of problems, and scholars within the field of computer science have developed various practical strategies to undertake descriptive analysis at multiple levels [21]. Addressing these two dimensions enhances the validity of the overall claims.

VI. Data, Evaluation, and Validity

Developing an understanding of data metrics and statistical analysis at the start of CS research facilitates a roadmap for acquiring data, selecting appropriate measurement methods, executing analysis, and anticipating potential threats to validity [1]. These same challenges may arise in the results chapter when planning the thesis report. Regardless of the chosen segment, predefined metrics, measurements, and statistical-inference strategies strengthen the overall validity of CS theses.

Identifying quality and reliability metrics constrains the contribution's characterization. During the implementation phase of software-engineering experiments, simple metrics must appear sufficient to exhibit changes and address potential challenges, including introducing extra complexity that may obscure the results. Establishing a framework for the expected contribution defines the requisite measurements and analyses before commencing the actual implementation.

6.1. Metrics, Measurements, and Statistical Inference

Conducting an exploratory case study using an industrial Haskell program, a broader spectrum of Haskell metrics is examined to clarify connections among the various modes of operation found in functional programs, assert the consistency and relevance of the previously investigated suite of textual metrics, and offer insights into the conjectured dark universe of software evolution. Measurement is paramount because quantifying either the wrong attributes or the correct ones inappropriately generates a superficially trustworthy outcome that either fails to describe the intended phenomenon or misrepresents it. All inquiries engage in a search for cause-effect relations, consult with a predetermined populace, and concentrate on the characteristics of selected members therein. The instruction is intended to assist scholars in substituting their chosen theme with a research question that remains suitably circumscribed in terms of both breadth and temporal-contextual specificity whilst declaring the involved actors. Articulating drafts and soliciting opinions from colleagues typically yield improvements in clarity and precision.

6.2. Threats to Validity and Mitigation

The design's focus on data collection and evaluations introduces specific threats to validity, which are usually not addressed when discussing validity in computer science studies. To enable clear understanding and independent replication, the data and all collected measurements are documented in the thesis [9]. Each measure is assigned to a corresponding metric, consistent with project and community norms. Numerous measurements are typically recorded to allow for more robust assessments and support multiple analyses, such as comparisons over time or between development steps. Since projects often generate large amounts of data over substantial periods, the degree of freedom in interpreting the data must also be recorded. Documents describing the evaluation processes of relevant prototype implementations accompany the thesis in the technical report. These documents specify the evaluation criteria used to access the prototypes according to the relevant project goals, detailing which criteria are met, to what degree, and which are excluded.

Benchmarking studies seek to assess the performance of competing systems or approaches within the same domain. Providing adequate justification for the selection of data sets and associated context is essential to establishing generality claims. Data sets used to benchmark the prototype implementations and availability information on corresponding data set generators are specified in the technical report. The benchmarks included in the evaluations were selected according to criteria that are explained. Only publicly accessible data sets available without contractual or financial constraints were chosen, ensuring compliance with community guidelines advocating for reproducibility.

VII. Ethics, Reproducibility, and Open Science

Inevitably, research raises ethical questions regarding human subjects, ecological impact, and ownership of others' ideas. Researchers are bound by several institutional, national, and even global guidelines. Studies involving humans or animals often require ethical approval, shaping the nature of experimental work, materials, and methods. Granting agencies may also impose restrictions, requiring analyses of negative consequences, harm, or potential misuse. Computer science typically relies on experimentation rather than traditional surveys or social studies. Researchers remain responsible for adhering to guidelines concerning any ethically sensitive aspect and declaring the relevant status to the audience.

Research in the natural and exact sciences is partly characterized by its reproducibility. Computational practices, methods, and output are documented through such media as code, documentation, electronic notebooks, papers, and data. Initially affected by parallel and distributed systems, reproducibility has further been compromised by technical limitations, documentation quality, surprises stemming from seemingly small architectural changes, and implementation provenance obscured by frame-dependent variability. A panoply of interdisciplinary approaches—including open-access articles, research objects, model repositories, virtual machines, executable papers, level-of-detail recording, and work-in-progress papers—seek to increase transparency and facilitate widespread experimentation. Many disciplines employ simulated experiments; methods exist to annotate experimental fact, procedure, and result; and automatic code generation, replication through spreadsheet-aware software, and combinatorial code search aim to mitigate barriers to commencement, repetition, and dissemination of computational experimentation. As computational research can be unintentionally nonreproducible, implementing the corresponding procedures constitutes good scientific practice.

7.1. Ethical Considerations in CS Research

Research ethics, a key concern in many fields, deserves careful consideration within computer science. Ethical dilemmas arise as research expands into sensitive areas such as machine learning and data mining, but even conventional studies warrant attention. Computer science—distinct among practical disciplines—comprises experimental, theoretical, and formal methods, with only the former invoking ethical reviews involving human subjects [22]. Nevertheless, broader ethical questions influence the scientific enterprise and shape the study of computer-science problems.

Research in computer science, like work in mathematics, perpetuates the essential cycle of inquiry, discovery, and validation. Each discipline, however, maintains unique conventions governing the pursuit of fundamental questions. To investigate ethical considerations in computer-science research rigorously, it is vital to characterize these conventions and relate them to the wider epistemic and social dimensions of the subject. Grounding this analysis in the author’s research into the mathematical theory of computing illuminates issues that arise in more theoretically oriented work.

7.2. Reproducibility Practices and Documentation

Reproducibility practices and documentation are essential for the integrity of science. Reproducibility allows a scientist to independently recreate a study and verify its results. A study is reproducible if it is possible to repeat it using the original data, methods, and analysis, and obtain comparable results [16]. It is a necessary condition for a study’s findings to become generally accepted. However, in many fields the reproducibility of computational solutions tends to be low. A common observation is that reproducing published works requires significant background knowledge and effort. Reproducibility is also affected by the publication process. The scrutinized nature of submitted manuscripts and code often weakens underpinning guarantees.

A scientist can face different situations when trying to repeat previous work. They may wish to reproduce a report using the same data set, methods, and analysis; they might want to replicate a study to confirm its results using an amended subset of the data or a distinct analysis; or they could intend to extend a study with contributions appealing to additional knowledge learned throughout the investigation [17]. It is thus important to ensure that both results and the underlying research remain reproducible. Reproducibility allows other researchers to verify the study and any associated propositions regarding its contribution or potential transferability to their context. It is therefore vital to entirely document both the study and the underlying research.

7.3. Open Data, Open Source, and Licensing

Open data, open source, and licensing are important for open science in general, and reproducibility aims at making research results accessible for repeat experiments and verifications. In the context of computer science, injunctions abound calling for data and source code to be disclosed together with scientific publications. The proposition rests on the premise that complete knowledge of the implementation is necessary to exhaustively understand the resulting software and is widely disseminated. Until recently, disclosure was deemed best practice, yet computer science remains largely an unsupported discipline. Interest in sharing software would grow only if community expectations demanded it. An unpublished study estimated an effective upper limit of 10 % on disclosed implementations. Poor adherence to such requirements is consistent across disciplines; much information deemed non-essential fails to be communicated. Consequently, any advice should prioritise the degree of adherence necessary to achieve the most impactful contribution [23]. Far from being pursued only out of generosity, sharing data sets or software frequently enhances the relevance of a study, especially when procedures or actions affect their dynamics.

A project can be labelled open, free, or public if a user is free to use, study, modify, or share it; compatible open-source licenses safeguard this right. The concept, borrowed from free software, historically included public-domain models allowing reappropriation without reference to the origin. Open and free options may converge as significant users begin to encourage sharing publicly, and as the absence of restrictions sustains a project sufficiently long for reputation-building. Individuals, even from the same lab, often disagree over such values and models, sometimes avoiding related regrets by abstracting the process rather than hard-coding it. Compatibility between open data and proprietary software illustrates the complexity of the landscape, especially where an expensive vacuum-cleaning robot monitors humidity. Selected options should therefore stem from specific motivations. Academics worrying about future investments might pursue economically sound projects with no large-scale competitive free alternative. Open-source software bundled with publications or isolated laboratories further fuels trade-off considerations [24].

VIII. Writing, Organization, and Thesis Structure

A research thesis is more than a set of scientific contributions; it conveys a coherent narrative that integrates these contributions within a reasonable premise. Striving for such coherence fosters engagement with the thesis and deepens appreciation of its value. Moreover, the thesis serves as the final examination for the graduate degree, and its organisation greatly influences the perception of quality. However, the structuring of this document exhibits a great variability across academic disciplines, even within the field of Computer Science.

Computer Science seeks to create knowledge, both theoretical and applied, through several questions and approaches. Consequently, the Foundations of Information and Computation combine large transdisciplinary topics that may be unique to the field. Drawing from experience in guiding students towards their graduation, the following description of a possible writing organisation proposes a coherent morphology that highlights Computer Science and its nature as the Science of Algorithms. Using the word "science" purposely not only for historic reasons, but also to help understanding by visibility, the thesis aiming to a solid and reachable graduation coverage split into the parts that shape the description, presentation and demonstration of the scientific findings in Computer Science outlines the theoretical and practical questions, the operation covering the foundations of the literature review and analysis of the scientific work, a part of computer science scientific method that can precede or follow the practical description, the experimental side underlining a request for implementation together with evaluation and validation, and finally a theoretical dimension referring to algorithmics and complexity. The computation organization thus shapes the manner picked for addressing the scientific subject of the thesis, tracing the discipline of Computer Science tackled throughout research.

8.1. Crafting a Coherent Narrative

The order and interplay of chapters, sections, appendices, and supplementary material strongly influence the coherence of a computer science thesis, determining both narrative flow and how well the required contributions engage with the central research question. Thesis readers increasingly expect the six-component structure of the problem-context-solution-contribution approach to be expressed within a cohesive narrative. Central to an effective narrative is a clear statement of the knowledge contribution that reflects the distinct claims made by the research [1]. Engaging broader computational and societal contexts aids in formulating such a contribution, but illuminating generic yet neglected aspects of the topic also proves helpful.

Addressing computer science-specific aspects of narrative construction early in the writing process better informs articulation of the knowledge contribution [25] and facilitates the communication of empirical evaluation plans and design- or implementation-centered research questions. Bookending the first chapter with an overview of the planned research design and a description of the overall thesis structure promotes a shared understanding of how core components relate to the central claim. Another conventional choice is creating a separate "Related Work" chapter focused on synthesizing prior approaches and clarifying how the proposed solution extends or combines them.

8.2. Appendices, Artifacts, and Supplementary Material

Supplementary material in a scientific thesis allows the author to add value to the work as a whole: for instance, supporting appendices or web links can lead the reader to supplementary material still being developed. Fully functional software artifacts, dataset collections, or traces of case studies can never be fully integrated into the narrative of a thesis without affecting its coherence. Furthermore, the vast amounts of effort required to develop and evaluate software artifacts, construct benchmark dataset collections, or perform system-sponsored case studies often exceed available time, and the desire to produce an artifact that is fully integrated into the thesis is likely to lead to disappointing results. Also, throughout the thesis and in the two accompanying papers, the emphasis is placed on careful documentation of work carried out in different research directions, along with detailed explanations of design and implementation decisions. Integrating this description into a single, coherent document is not feasible. A zip file is made available containing sufficient details for reproducing them under the Apache 2.0 license. Analysis of the collected datasets indicates that reproducible research in the solid state chemistry domain remains a challenge, a finding worth reporting. A trace of a case study of a software engineering project carried out at a large institution offers lessons for teaching the software engineering process; an overview of that extent is likely useful to other educators. Finally, a publication is shared as its main finding confirms the importance of this area of inquiry [1].

IX. Impact, Implications, and Future Work

When phrasing the knowledge contribution, avoid judgmental phrasing like “informed decision,” “well-defined,” or “well-studied.” The remaining questions can be incorporated in as comfortable a way as possible, possibly as part of the overview.

Engagement with the reader is mostly absent in the references; include a few reader-oriented phrases, e.g., “the following outlines” or “the study has sought to.” [1] Consolidating the discussion of practical relevance with the opening paragraph may also strengthen the connection to the contributions already made. Individual limitations can also be handled in a more engaging manner.

The prospects for research extension are not explicitly delineated in the references; if this detail proves appropriate for the audience, include a phrase directing attention to this facet of the work.

The discussion of real-world uptake presently lacks further specificity; augment this portion with concrete examples or domains that would furnish a more tangible sense of application.

In the discussion of research extension, references to theory should specify whether the emphasis lies on algorithmic strategies, proof techniques, complexity characterization, or hardness results. The “real-world” dimension should indicate whether it pertains to industrial application, governmental operation, societal impact, or another facet.

When defining the study’s focus, frame it as a contrast with situations presenting both diverse user populations and flexible adaptation. Use active constructions such as “studies” instead of “the work,” and fine-tune the tone further to align more closely with the cited literature.

9.1. Knowledge Contribution and Practical Relevance

In the course of a CS thesis, it is essential to articulate the contribution to knowledge clearly, as this defines the thesis's scope and demonstrates its relevance. The contribution is especially important for companies, partners, and practitioners who expect clearly defined research questions as well as knowledge contributions that can subsequently be applied in real-world settings [1]. Framing the scientific impact is thus often a challenging yet crucial undertaking, but CS students and researchers specifically often grapple with how to tackle this aspect effectively.

The knowledge contribution needs to be stated as explicitly as possible and positioned with respect to the existing body of knowledge; the contribution is often strengthened by spelling out technical or methodological innovations and their practical implications. For students who are not familiar with explicitly communicating knowledge contributions, several avenues are available for remediation. Framing the contributions according to established contributions models and frameworks (e.g., according to the taxonomy proposed by Oates, 2006) can help structure thinking on these issues and encourage further iterative reflection on established contributions, the underlying arguments, and available literature [26].

9.2. Limitations and Paths for Extension

The contribution of the current research is twofold: (1) the development of a software system to facilitate the synchronization of data acquired from heterogeneous IoT sources, and (2) the design of an algorithm to estimate a user’s travel time between invertible points—a novel addition to the literature on travel time estimation. The proposed solution enables synchronization of a wide variety of IoT devices and formats—even those using proprietary protocols. As the Internet of Things (IoT) gains traction, this capability is essential for constructing smart-city test beds that require correlated streams of information over extended time spans [1]. An implementation of the stated requirements is available in an online repository. The extension directly supports urban mobility models, while the incorporation of temporal analysis functions and clustering algorithms eases exploration of a complex travel-diary data set.

The current research has implications in at least two additional domains: (1) traffic-generation models and (2) crowdsourcing. Traffic-generation models are indispensable for experimenting in smart-city simulation environments; the proposed estimate-travel-time algorithm constitutes a vital step in this context [4]. The introduction of travel diaries permits exploration of data not previously available in microscopic transportation models. The capability to synchronize data gathered from heterogeneous IoT devices opens opportunities for developing new public collaborative-crowdsourcing solutions beyond those implemented in existing applications.

X. Conclusion

Technological advances have made software pervasive across many sectors and applications. Consequently, the associated fields of software engineering and computer science continue growing at a fast pace, demanding reliable methods for analysis, design, verification, and validation of software. These same advances have made software engineering and computer science more accessible to a broader range of people, increasing the number of students conducting master's theses in those fields. Additionally, computer science remains one of the least female-represented fields in academia and the industry [4]. Given the researcher’s own experience in industry and academy, alongside the gender perspective, the researcher provides guidance on the research methodology employed in an illustrative software and computer science master's thesis. The aim is to contribute broader insights that can assist other graduates in the computing disciplines.

Conflict of Interest
The author declares no conflict of interest

REFERENCES

[1] E. Knauss, "Constructive Master's Thesis Work in Industry: Guidelines for Applying Design Science Research," 2020.
[2] N. Vasilyeva, "How do students know they are right and how does one research it?," 2018.
[3] B. Kei Daniel, M. Asil, and S. Carr, "Psychometric properties of the TACT framework—Determining rigor in qualitative research," 2024.
[4] S. B. Adams, "Gaming and computer science: Bridging the gender-gap," 2019.
[5] D. Remenyi, "So you want to be an academic researcher in business and management studies!," 2007.
[6] D. Burkart, "Demarcating Computer Science," 2012.
[7] J. Jawitz, J. Case, and D. Marshall, "Grappling with methodologies in educational research: science and engineering educators finding their way," 2009.
[8] E. Clarke and J. Visser, "Pragmatic research methodology in education: possibilities and pitfalls," 2018.
[9] M. Wyrich and S. Apel, "Evidence Tetris in the Pixelated World of Validity Threats," 2024.
[10] P. Ralph, M. Kuutila, H. Arif, and B. Ayoola, "Teaching Software Metrology: The Science of Measurement for Software Engineering," 2024.
[11] C. Costley and P. Nottingham, "Revisiting search and review for work-based projects," 2018.
[12] D. Falchetti, "The Novelty Journey in Evaluation Processes: The Role of Personal Traits, Social Factors and Idea Framing in Shaping Audience Preferences," 2018.
[13] M. Felderer and G. Horta Travassos, "The Evolution of Empirical Methods in Software Engineering," 2019.
[14] L. M. Alves, R. J. Machado, and P. Ribeiro, "Experimental software engineering in educational context," 2012.
[15] T. F. M. Sirqueira, M. A. Miguel, H. L. O. Dalpra, M. A. P. Araujo et al., "Application of Statistical Methods in Software Engineering: Theory and Practice," 2020.
[16] S. Arabas, M. R. Bareford, L. R. de Silva, I. P. Gent et al., "Case Studies and Challenges in Reproducibility in the Computational Sciences," 2014.
[17] T. Crick, "Reproducibility in Research: Systems, Infrastructure, Culture," 2017.
[18] M. H. ter Beek and A. Ferrari, "Empirical Formal Methods: Guidelines for Performing Empirical Studies on Formal Methods," 2022.
[19] K. A Gilpatric, "Complexity: Theoretical and methodological applications for sociology," 2001.
[20] L. da F. Costa and G. S. Domingues, "Cost-Based Approach to Complexity: A Common Denominator?," 2019.
[21] A. R. S. Araujo, H. Perrelli de Moura, and C. França, "COMPLEXITY WITHIN SOFTWARE DEVELOPMENT PROJECTS: AN EXPLORATORY OVERVIEW," 2015.
[22] D. R. Wright, "Motivation, Design, and Ubiquity: A Discussion of Research Ethics and Computer Science," 2007.
[23] J. M. Pearce, "Open Source Research in Sustainability," 2012.
[24] A. Guadamuz González, "Open science: open source licences for scientific research," 2006.
[25] L. McAlpine, "Miks kasutada narratiivi? Lugu narratiivist," 2016.
[26] W. Presthus and B. Erik Munkvold, "How to frame your contribution to knowledge? A guide for junior researchers in information systems," 2016.