

Decision Visibility and Product Thinking in Software Engineering Education: A Conceptual Model of Quality-Oriented Reasoning

Tamara ZIVKOVIC*

Faculty of Informatics and Computing, Singidunum University, Danijelova 32, 11000 Belgrade, Serbia

e-mail: tzivkovic@singidunum.ac.rs

Abstract. Persistent software quality problems continue to occur despite advances in software engineering methods, tools, and standards. This paper proposes a conceptual explanation suggesting that such problems may partly reflect how software engineers are educated to reason about quality-relevant decisions under uncertainty. The study introduces decision-oriented product thinking as an engineering competence concerned with trade-offs, prioritization, value-sensitive reasoning, and long-term quality consequences. Drawing on software engineering education research, industry skills-gap studies, and state-of-practice investigations in requirements engineering, the paper develops a conceptual model linking educational framing, decision visibility, product-thinking practices, and quality-oriented reasoning. The analysis identifies an opportunity to integrate existing forms of decision-oriented learning more systematically across software engineering education. Although contemporary curricula and educational practices already address elements of decision reasoning, quality-relevant choices, assumptions, alternatives, and trade-offs are not necessarily made consistently visible as objects of instruction, reflection, and assessment. The paper proposes decision visibility as an integrative educational mechanism for connecting these existing elements with product-thinking practices and quality-oriented reasoning. The contribution of the study is conceptual and explanatory. In addition to proposing a conceptual model, the paper introduces educational design principles and classroom-oriented applications intended to make quality-relevant decisions more visible during software engineering education.

Key words: Software engineering education; Decision visibility; Product thinking; Quality-oriented reasoning; Software quality; Requirements engineering.

1. Introduction

Despite decades of progress in software engineering methods, tools, and standards, software quality remains a persistent and widespread challenge. Defects, rework, misaligned functionality, and systems that fail to deliver expected value continue to appear in all domains, including organizations with mature development processes. Such quality problems are commonly attributed to technical causes, including insufficient testing, inadequate or unstable requirements, architectural debt, and process failures Garousi *et al.*

*Corresponding author.

(2019); Oguz and Oguz (2019). While these explanations capture important aspects of quality failure, they mainly describe where quality breaks down rather than why similar failures occur across projects, organizations, and contexts.

This paper examines a complementary educational perspective on software quality: how explicitly software engineering education makes quality-relevant decisions visible for reasoning, discussion, and assessment. Software engineers in professional practice routinely reason about requirements, competing quality attributes, delivery constraints, uncertainty, and longer-term consequences Franch *et al.* (2023). Software engineering education already includes decision-oriented elements through design activities, stakeholder engagement, project work, and quality-related learning outcomes; however, the visibility of the choices, assumptions, alternatives, trade-offs, and rationales involved may vary across learning activities.

Research on the education-industry gap documents mismatches in skills, tools, professional competencies, and exposure to real-world constraints Daun *et al.* (2023); Cico *et al.* (2021); Akdur (2022); Diniz *et al.* (2024); Mittas *et al.* (2025). The perspective developed here complements this literature by focusing not on the general absence of decision-making education, but on how educational framing makes quality-relevant reasoning explicit and assessable. The proposed relationship between such educational framing and later professional reasoning is conceptual rather than an empirically demonstrated effect.

Within this perspective, product thinking is treated as a decision-oriented engineering competence for reasoning about software as a value-bearing system situated within technical, user, and organizational contexts. It concerns identifying quality-relevant choices, evaluating trade-offs under uncertainty, and articulating their rationale and anticipated consequences. Product thinking is therefore used here as an engineering lens rather than as a synonym for product management or as a replacement for technical competence.

Building on existing decision-oriented elements in software engineering education, this paper conceptualizes decision visibility as an educational dimension connecting educational framing with opportunities to practice product-thinking and quality-oriented reasoning. The contribution is integrative rather than based on identifying an entirely new competence: the model focuses on whether quality-relevant choices, assumptions, alternatives, trade-offs, and rationales are made explicit and assessable. The proposed relationships are theoretical and do not establish that educational framing causes later professional behavior or software quality outcomes.

Accordingly, the study addresses the following research question: *How might educational framings in software engineering influence the visibility of quality-relevant decisions in ways that provide opportunities to develop product-thinking practices and quality-oriented reasoning?* The main contributions are:

- A conceptual model linking educational framing, decision visibility, product-thinking practices, and quality-oriented reasoning.
- An operational conceptualization of product thinking as a decision-oriented engineering competence distinct from product management and adjacent constructs.
- A set of theoretical propositions and educational design principles for making quality-relevant choices, trade-offs, assumptions, and rationales more visible in learning and assessment.

- Classroom-oriented applications illustrating how the proposed model can inform software engineering teaching and assessment without presuming empirically demonstrated effects.

The remainder of the paper is organized as follows. Section 2 discusses adjacent constructs and positions product thinking within existing software engineering education research. Section 4 outlines the conceptual and methodological approach. Section 3 presents the central conceptual model, including the construct architecture, decision-visibility mechanism, and conceptual propositions. Sections 5 and 6 examine how quality-relevant decision-making is represented across software engineering education and contrast these educational framings with decision-making in industrial software engineering practice. Section 8 presents educational design principles and classroom applications derived from the model. Section 9 discusses boundary conditions, rival explanations, and implications for future research. The paper concludes by positioning product thinking as a teachable form of quality-oriented engineering reasoning.

2. Background and Adjacent Constructs

This section positions the present study within existing research on software engineering education, software quality, and professional reasoning in software engineering practice. In addition to reviewing prior work on education-industry alignment and quality-related decision-making, the section distinguishes decision-oriented product thinking from adjacent constructs including requirements engineering, design rationale, technical debt awareness, product management, and generic professional skills. This positioning is necessary because the present study does not introduce product thinking as a replacement for these constructs, but as an integrative educational lens focused on the visibility and justification of quality-relevant decisions.

2.1. Education-Industry Alignment and Skills Gap

A substantial body of research examines alignment between software engineering education and industrial needs. Systematic reviews, mapping studies, and surveys identify mismatches involving tools, practices, professional competencies, requirements engineering, communication, and practical experience Garousi *et al.* (2019); Cico *et al.* (2021); Daun *et al.* (2023); Akdur (2022); Mittas *et al.* (2025); Metrôlho *et al.* (2022); Diniz *et al.* (2024).

Related work also examines technology mismatches, industry-academia collaboration, and the realism of educational and research settings Dobslaw *et al.* (2023); Wohlin and Runeson (2021); Song and Runeson (2023); Marijan and Gotlieb (2021). These studies establish a broader alignment problem but do not by themselves demonstrate a general absence of decision-making education. The present study therefore uses this literature as context for examining the more specific question of how quality-relevant decisions and their rationales are made visible in software engineering learning activities.

2.2. *Software Quality and Requirements Engineering in Practice*

Research on requirements engineering shows that early lifecycle decisions can have important implications for long-term software quality. Large-scale interview studies describe requirements elicitation and specification as involving uncertainty, negotiation, and informal trade-offs rather than consistently stable and well-defined inputs Franch *et al.* (2023); Fernández *et al.* (2017). Studies of standards usage in industry further show that formal models coexist with pragmatic adaptation and selective compliance Franch *et al.* (2021).

Recent work has extended this perspective to emerging domains such as AI-based systems, where requirements maturity is constrained by uncertainty and evolving stakeholder expectations Habiba *et al.* (2024). These findings reinforce the view that software quality is not solely a result of downstream verification but is shaped by upstream prioritization and interpretation decisions.

Research on prioritization and planning supports this interpretation. Systematic literature reviews document a wide range of techniques for requirements and task prioritization, but also note limited empirical guidance on how trade-offs are resolved in practice Bukhsh *et al.* (2020); Bugayenko *et al.* (2023). Similarly, studies on test case selection and prioritization focus on optimization methods without fully addressing the decision rationale underlying quality trade-offs Pan *et al.* (2022).

2.3. *Decision-Making and Trade-offs in Software Engineering*

Decision-making has been examined in several specialized subfields of software engineering. Behavioral software engineering research highlights the socio-technical nature of engineering work and the role of human judgment in shaping outcomes Lenberg *et al.* (2015). Early work on software engineering decision support emphasized the need for structured reasoning under uncertainty Ruhe (2002), while more recent contributions articulate design principles for improving software design decisions Tang and Kazman (2021).

Architectural decision-making has received growing attention, particularly in relation to quality attribute trade-offs. Techniques for explaining and visualizing trade-off spaces illustrate how design choices constrain achievable quality levels Cámara *et al.* (2023a,b); Öqvist *et al.* (2024). Research on technical debt further demonstrates how deliberate short-term decisions accumulate into long-term quality consequences Junior and Travassos (2022); de Toledo *et al.* (2021); Kotti *et al.* (2023).

Table 1 summarizes how decision-oriented product thinking relates to adjacent constructs discussed in software engineering education and practice.

2.4. *Educational Treatment of Decision-Making*

Within software engineering education research, decision-related reasoning appears in multiple forms, including project-based learning, stakeholder interaction, requirements activities, and broader professional competencies. Reviews of requirements engineering education document substantial use of authentic and experiential teaching approaches

Table 1
Product thinking and adjacent constructs

Adjacent construct	What it covers	What it may miss	Product-thinking contribution
Requirements engineering	Elicitation, specification, validation, and requirements artifacts.	May treat requirements primarily as documents rather than as value-sensitive decisions.	Emphasizes prioritization, stakeholder value, uncertainty, and quality-related trade-offs.
Design rationale	Explanation and documentation of design choices and their justification.	May focus on the rationale for particular decisions without specifying how decision reasoning is made visible across learning activities.	Uses rationale as one form of evidence within a broader educational focus on visible choices, alternatives, assumptions, trade-offs, and consequences.
Technical debt awareness	Reasoning about the consequences of short-term choices and deferred work.	Addresses an important class of long-term engineering decisions but does not by itself provide a general framework for making decision reasoning visible across educational activities.	Connects technical-debt decisions with a broader educational focus on prioritization, trade-offs, rationale, and quality consequences.
Product management	Market positioning, user needs, business priorities, and product ownership.	Often treated as external to software engineering education and technical practice.	Reframes product-oriented reasoning as part of engineering competence rather than only managerial responsibility.
Professional skills	Communication, teamwork, leadership, and employability.	May remain generic and insufficiently connected to software engineering decisions.	Positions decision-making as a discipline-specific engineering competence grounded in trade-offs, uncertainty, and software quality reasoning.

alongside instruction in techniques, models, and documentation practices Daun *et al.* (2023); Cico *et al.* (2021). However, the extent to which the quality-relevant decisions embedded in these activities are explicitly identified, justified, and assessed varies across educational settings. Skills gap studies frequently identify deficiencies in critical thinking and contextual understanding but conceptualize these primarily as transversal skills rather than as decision-oriented competencies Akdur (2022); Mittas *et al.* (2025).

A substantial body of educational research focuses on improving technical competence and quality assurance through learning environments and automated analysis tools. Surveys of software testing education highlight the central role of structured laboratory environments and tooling in supporting student learning Živkovic and Živkovic (2021). Comparative studies of testing techniques further emphasize method selection and technical effectiveness rather than the rationale underlying quality-related decisions Živković and Živković (2020).

Recent work on defect prediction and quality analytics introduces advanced machine learning and optimization techniques to support software quality assessment Živkovic *et al.* (2023b); Villoth *et al.* (2025). These approaches strengthen predictive and diagnostic capabilities, but they primarily frame quality as a measurable outcome rather than as the result of cumulative design and prioritization decisions.

Research on software testing education also documents the use of learning environments and tool-supported approaches for organizing and supporting testing activities Živkovic *et al.* (2023a). More recent intervention-based work reports improvements associated with automated feedback and quality monitoring tools in specific educational settings Nocera *et al.* (2025). These approaches demonstrate the educational value of tool-supported quality assessment, while their primary focus remains on feedback, analysis, and artifact quality rather than on making students' reasoning about uncertainty, trade-offs, and long-term quality consequences an explicit object of instruction and assessment.

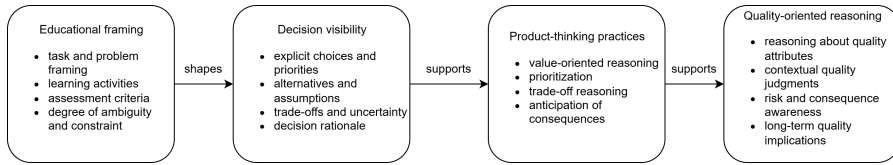


Figure 1. Overview of the proposed relationship between educational framing, decision visibility, product-thinking practices, and quality-oriented reasoning.

Together, these strands show that software engineering education addresses quality through multiple technical, experiential, and tool-supported approaches. The specific question pursued in this paper is how these existing approaches can also make students' reasoning about quality-relevant choices, uncertainty, and trade-offs more explicit for instruction and assessment.

2.5. *Why Product Thinking Requires a Distinct Educational Framing*

Taken together, existing research provides substantial insight into education-industry misalignment, requirements practices, software quality trade-offs, and socio-technical decision-making in software engineering. However, these strands of literature remain conceptually fragmented. Education-focused studies primarily emphasize competencies, tools, and curricular content, whereas software quality research tends to focus on processes, metrics, architectures, or technical artifacts. Decision-making appears across these domains in different forms, while its explicit treatment as an educational object varies across the literature and educational contexts considered here.

The present study differs from prior work by conceptualizing decision visibility as a proposed link between educational framing and opportunities to practice quality-oriented reasoning. Rather than proposing additional software engineering topics or professional skills, the paper examines how software engineering education may shape which quality-relevant decisions students learn to recognize, justify, and evaluate. In this sense, product thinking is positioned as a decision-oriented engineering competence that integrates technical reasoning with value-oriented judgment under uncertainty.

This perspective does not replace requirements engineering, design rationale, technical debt education, or product management perspectives. Instead, it integrates them through a common educational concern: whether students are explicitly required to recognize and justify software-quality decisions under realistic constraints, ambiguity, and competing stakeholder priorities.

Table 2 summarizes key contrasts between educational and industrial framings, while Figure 1 provides an overview of the proposed conceptual relationships developed in the following section. It is intended as a conceptual comparison rather than as a measured or universally representative description of software engineering education or industrial practice.

Building on prior education-industry alignment research, the present study focuses specifically on the educational visibility of software-quality decisions. The conceptual

Table 2
Illustrative contrast between educational and industrial framings synthesized from the reviewed literature

Dimension	Illustrative educational framing	Illustrative industrial framing
Role of requirements	May be presented as relatively stable inputs to be analyzed and specified	Treated as evolving and negotiable commitments shaped by stakeholders and constraints
View of software quality	Framed as compliance with specifications and correctness of artifacts	Framed as the outcome of prioritization and trade-offs among competing quality attributes
Handling of uncertainty	Problems may be well-defined and relevant information may be substantially specified	Decisions are made under incomplete, ambiguous, and changing information
Treatment of trade-offs	Trade-offs may remain implicit or may be addressed within particular tasks and learning activities	Trade-offs may require balancing technical, human, and organizational concerns
Responsibility for decisions	Responsibility is diffused across methods, processes, and prescribed roles	Responsibility is implicit and often borne by engineers through everyday choices
Timing of quality shaping	Quality may be evaluated primarily through implementation, testing, and verification	Quality is shaped early through decisions about scope, priorities, and acceptable risk

contribution lies not in identifying an entirely new competence, but in proposing decision visibility as an integrative link between educational framing and opportunities to practice product-thinking and quality-oriented reasoning.

3. The Conceptual Model of Decision-Oriented Product Thinking

This section presents the central conceptual model developed in this study. The model proposes relationships among educational framing, decision visibility, product-thinking practices, and quality-oriented reasoning. Building on the overview introduced earlier in Figure 1, the section operationalizes decision-oriented product thinking as an engineering competence, defines its conceptual components, and specifies how decision visibility is conceptualized within the model.

3.1. Decision-Oriented Product Thinking as an Engineering Competence

In this study, decision-oriented product thinking is defined as the capacity of software engineers and students to identify quality-relevant choices, evaluate competing stakeholder and technical constraints, justify trade-offs under uncertainty, and anticipate the long-term consequences of decisions for software value and software quality.

This conceptualization positions product thinking as a form of engineering reasoning rather than as a managerial role or a generic professional skill. The emphasis is placed on how engineers recognize, interpret, justify, and communicate decisions that influence software quality over time.

Table 3 summarizes the conceptual architecture of decision-oriented product thinking and its observable educational indicators.

The examples in Table 3 are illustrative rather than empirical student data. They are included to demonstrate how the proposed components and educational indicators could become observable in student reasoning and artifacts.

Table 3
Components of decision-oriented product thinking

Component	Definition	Educational indicator	Illustrative example of student work
Value orientation	Connecting technical decisions to user, stakeholder, or organizational value.	Student identifies who benefits or is affected by a design or quality decision.	"We prioritized accessibility over feature X because it affects core users."
Trade-off reasoning	Comparing competing quality attributes and constraints.	Student explains why one quality attribute is prioritized over another under current constraints.	"We accepted lower performance because maintainability was more critical at this stage."
Uncertainty handling	Making provisional decisions under incomplete or evolving information.	Student articulates assumptions, risks, and conditions that could change the decision.	"The requirement is unclear; we assume local storage, but this creates privacy risks."
Long-term quality reasoning	Anticipating lifecycle, maintenance, technical debt, and future-change implications.	Student connects present implementation choices to future quality consequences.	"This shortcut accelerates delivery but increases future integration risk."
Decision accountability	Documenting alternatives, rejected options, and explicit rationale.	Student produces decision logs, rationale explanations, or prioritization justifications.	"We rejected option B because it increased coupling and reduced testability."

3.2. *Software Quality as an Outcome of Decisions, Not Only Processes*

Software quality is commonly evaluated through measurable indicators such as defect density, performance metrics, or compliance with requirements and standards Garousi *et al.* (2019). Although such indicators are useful, they primarily capture the outcomes of development activities rather than the mechanisms through which quality is produced. In practice, software quality can be understood as emerging from a sequence of decisions made throughout the software lifecycle, including decisions related to scope definition, prioritization, trade-offs, and acceptable levels of risk.

Quality-related problems may be influenced by early decisions made under uncertainty, including situations involving incomplete requirements, evolving stakeholder expectations, or constraints related to time and resources. Decisions about which functionality to implement, which quality attributes to emphasize, and which issues to postpone can have long-term consequences that may constrain what subsequent verification or testing activities can achieve. From this perspective, software quality can be viewed as being shaped throughout development by quality-relevant engineering decisions rather than only evaluated at the end of the process.

For example, prioritization decisions that defer non-functional requirements under schedule pressure may constrain later architectural choices and increase the difficulty of achieving particular quality objectives through subsequent testing or refactoring.

Software quality education can combine procedural and artifact-based approaches with decision-oriented learning. From the perspective developed here, the relevant issue is whether students are also required to make explicit the choices, assumptions, trade-offs, and rationales that shape quality-related outcomes. Making this reasoning visible provides opportunities to discuss and assess quality as an object of engineering judgment as well as technical verification.

3.3. Decision-Making as a Core Socio-Technical Competence

Decision-making in software engineering is inherently socio-technical Lenberg *et al.* (2015); Ruhe (2002). The components summarized in Table 3 illustrate how decision-oriented product thinking extends beyond procedural correctness toward explicit reasoning about value, trade-offs, uncertainty, and long-term software quality consequences. Engineers are required to balance technical constraints with human, organizational, and economic considerations, often in the absence of complete, stable, or unambiguous information. Such decisions involve judgment, prioritization, and negotiation and may have implications for the quality attributes that a system ultimately exhibits.

In industrial contexts, software engineers interact with product owners, designers, managers, and users and may contribute to decisions concerning requirements, technical debt, refactoring, and release timing. Decision-oriented reasoning competence should, however, be distinguished from the authority and organizational safety required to act on such reasoning. An engineer may be able to identify and justify a quality-relevant trade-off without having the formal authority, resources, or organizational support to implement the preferred choice. The present model concerns the development and visibility of reasoning competence rather than the distribution of decision authority in professional organizations.

Explicit decision reasoning already has precedents in software engineering education. For example, Jørgensen *et al.* describe an evidence-based software engineering approach in which students critically evaluate empirical evidence and use it to support software engineering decisions Jorgensen *et al.* (2005). This demonstrates that decision-oriented reasoning is not absent from existing software engineering education. The educational opportunity addressed in this paper is therefore not to introduce decision-making as an entirely new competence, but to make quality-relevant decisions more systematically visible across different learning activities, including the choices, alternatives, assumptions, trade-offs, and rationales through which software quality is shaped.

3.4. Product Thinking as a Decision-Oriented Perspective

In this paper, product thinking is conceptualized as a decision-oriented perspective (as an engineering competence) that enables engineers to reason about software as a value-bearing system situated within a broader context of use, evolution, and constraint. Product thinking emphasizes understanding why a system is built, for whom, and under what conditions, as well as how engineering decisions influence both immediate functionality and long-term quality. Figure 2 illustrates this positioning of product thinking as the intersection of technical correctness, user value, and organizational constraints.

Product thinking is not equated with product management roles or responsibilities. In the context of software engineering, it refers to structured reasoning about value, risk, trade-offs, and the longer-term quality implications of concrete engineering choices, including requirements prioritization, architectural decisions, and technical debt. It is therefore distinguished from both managerial product ownership and generic professional skills by its focus on quality-relevant engineering decisions under uncertainty.

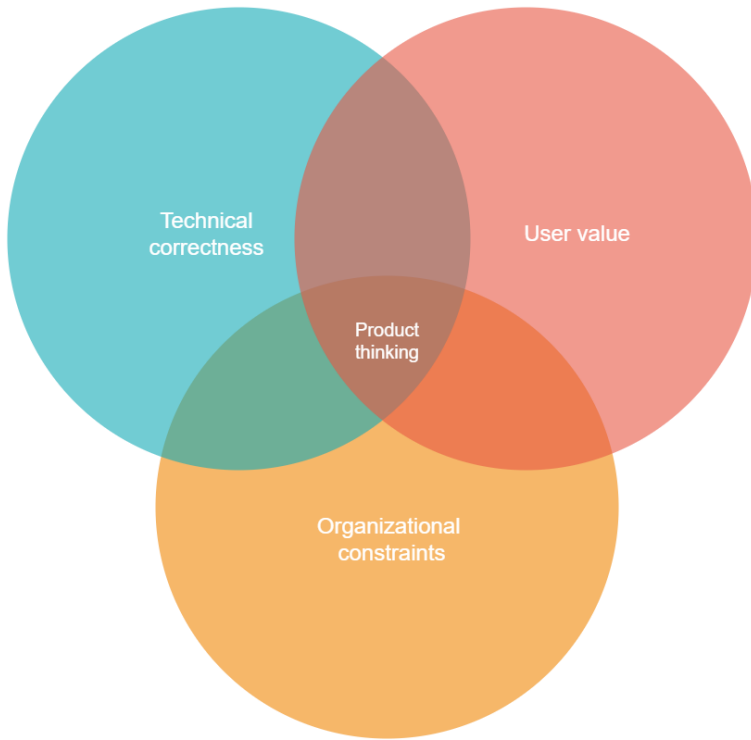


Figure 2. Product thinking as a decision-oriented engineering competence.

By foregrounding value and impact, product thinking provides a lens for making decision logic explicit in engineering education. It encourages students to examine requirements as potentially evolving and quality attributes as context-dependent priorities rather than treating either as fixed inputs. In this sense, product thinking complements formal engineering methods by foregrounding the reasoning behind quality-relevant choices.

In operational terms, product thinking can be examined through the observable indicators summarized in Table 3, including rationale articulation, trade-off reasoning, value orientation, uncertainty handling, and anticipation of longer-term quality consequences. These indicators provide a basis for future empirical investigation in educational and professional contexts.

3.5. *Decision Visibility as an Educational Mechanism*

Educational practices influence not only what students learn, but also which forms of decision-making become visible, discussable, and assessable during software engineering education Daun *et al.* (2023). When curricula prioritize predefined problem specifications, procedural correctness, and compliance-oriented assessment, students primarily encounter software development as task execution rather than as a process of negotiating quality-relevant decisions under uncertainty.

From this perspective, decision visibility refers to the extent to which educational environments explicitly expose students to trade-offs, prioritization, rationale articulation, uncertainty management, and the long-term consequences of engineering choices.

Decision visibility should be distinguished from design rationale. Design rationale concerns the content and documentation of reasons for particular design choices, whereas decision visibility concerns the educational conditions under which decisions and their associated reasoning become available for discussion, reflection, and assessment. A rationale is therefore one possible form of evidence through which a decision becomes visible, but decision visibility is the broader educational property: it also includes whether alternatives, assumptions, uncertainties, competing priorities, and consequences are explicitly surfaced during learning activities. The two concepts are complementary rather than interchangeable.

The extent to which students confront the consequences of their choices depends partly on whether decision reasoning is made explicit in learning activities and assessment. Coursework that emphasizes correctness and completeness can coexist with decision-oriented learning, but it may provide fewer opportunities for explicit reflection when the rationale behind choices, realistic constraints, and competing quality concerns are not themselves examined.

This paper therefore focuses on variation in decision visibility rather than on the general absence of decision-making from software engineering education. Figure 3 illustrates this distinction by contrasting elements that may be made explicit in educational settings with quality-relevant decision reasoning that may remain implicit. The conceptual concern is whether choices, assumptions, alternatives, trade-offs, and consequences become sufficiently visible to support discussion, reflection, and assessment.

The central mechanism proposed in this study can therefore be summarized as follows:

Educational framing → decision visibility → product-thinking practices → quality-oriented reasoning

The model does not propose direct causal relationships between educational experiences and software quality outcomes. Rather, decision visibility and product-thinking practices are treated as conceptual links through which educational framing may support quality-oriented reasoning relevant to professional software engineering practice. Whether, to what extent, and under what conditions these proposed relationships affect professional reasoning or software quality requires empirical investigation beyond the scope of the present conceptual study.

3.6. Conceptual Propositions

The conceptual model developed in this paper leads to five theoretical propositions that articulate relationships proposed for future empirical investigation. These propositions should be understood as theory-derived expectations and design assumptions rather than as empirically demonstrated effects of the present study.

P1: Educational framings that emphasize correctness and compliance without making quality-relevant choices explicit may provide fewer opportunities for students to examine value trade-offs as part of engineering responsibility.

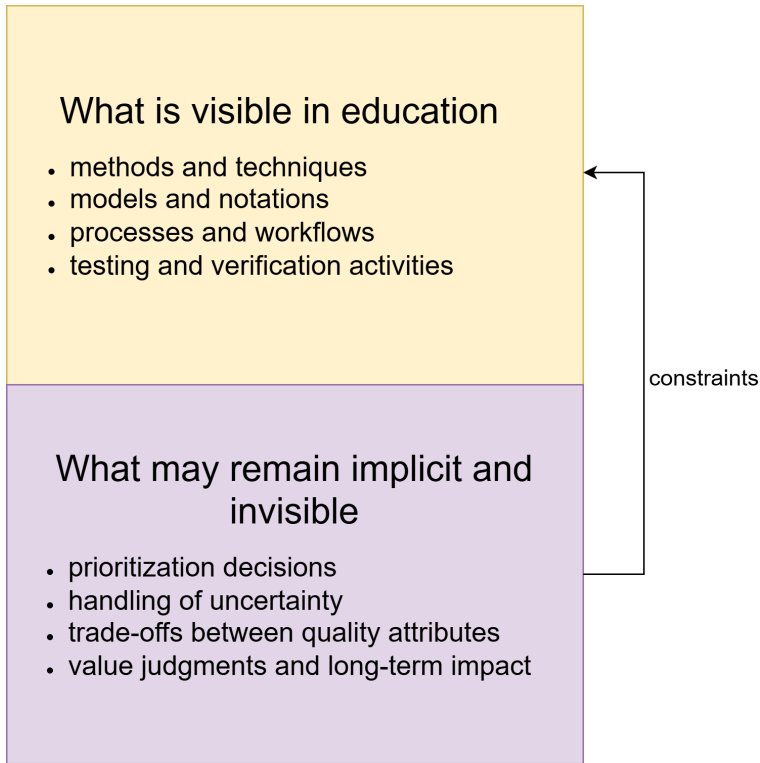


Figure 3. Decision visibility as the distinction between elements made explicit in software engineering education and quality-relevant decision reasoning that may remain implicit.

P2: Requiring students to justify decisions related to scope, prioritization, risk, and quality attributes creates opportunities to practice product-thinking competence.

P3: Decision visibility is proposed as a conceptual mechanism through which curriculum design may influence students' quality-oriented reasoning.

P4: Assessment practices that reward only technically correct artifacts may provide fewer opportunities to make product-thinking practices visible, whereas assessment of rationale can make trade-off reasoning more explicit and assessable.

P5: Product thinking is particularly relevant to learning situations involving ambiguous requirements, conflicting stakeholder goals, incomplete information, or long-term maintenance consequences.

3.7. Positioning Within Existing Research

The perspective developed in this section builds on research on education-industry alignment, skills gaps, and decision-oriented learning by using decision visibility as an integrative conceptual mechanism. This decision-oriented lens provides the basis for examining how existing forms of quality-relevant reasoning are made explicit, discussable, and

assessable across the educational and industrial literature considered in the following sections.

4. Methodological Approach

This study adopts a qualitative, integrative, and conceptual approach to reinterpret evidence from software engineering education and industrial practice through a decision-oriented analytical lens. It does not present new primary empirical data or seek to establish causal relationships between education and software quality outcomes. Narrative synthesis was used to support explanatory integration across heterogeneous educational and industrial literature rather than quantitative aggregation of findings.

4.1. Source Selection and Corpus Definition

The literature included in this conceptual synthesis was selected purposively rather than exhaustively. The objective was not to conduct a systematic review of all relevant publications, but to construct a theoretically informative corpus capable of supporting an integrative and explanatory argument.

The corpus was organized around four complementary domains of evidence:

1. software engineering and requirements engineering education research, particularly systematic reviews and mapping studies addressing curriculum design, competencies, and educational practices;
2. industry-oriented skills-gap and competency studies examining professional expectations and graduate preparedness in software engineering;
3. state-of-practice investigations in requirements engineering and software development that explicitly discuss decision-making, prioritization, uncertainty, and quality-related trade-offs in industrial contexts;
4. software engineering research addressing architectural decisions, technical debt, prioritization, and socio-technical reasoning related to software quality.

Priority was given to peer-reviewed studies that were either highly cited, large-scale, systematic in nature, or widely recognized within software engineering education and software engineering practice research. Sources were selected based on their relevance to the paper's central conceptual concern: how educational framing may influence professional reasoning related to software quality.

Table 4 summarizes the primary source domains and their role in the conceptual argument developed throughout the paper.

4.2. Analytical Strategy

The analysis was conducted in three stages. First, systematic reviews and mapping studies on software engineering and requirements engineering education were examined to

Table 4
Source domains and their contribution to the conceptual argument

Source domain	Main evidence used	Role in conceptual argument
Software engineering and computing education research	Systematic reviews, mapping studies, and curriculum-oriented research on software engineering education, requirements engineering education, and competency development	Identifies educational emphases and representations of decision-making and software quality
Industry-oriented skills-gap and competency studies	Empirical studies examining industrial expectations, graduate preparedness, and competency mismatches	Illustrates how professional expectations extend beyond technical execution toward contextual and decision-oriented reasoning
State-of-practice investigations in requirements engineering and software development	Interview studies and industrial investigations addressing requirements practices, prioritization, uncertainty, and quality trade-offs	Illustrates quality-relevant decision-making under uncertainty in professional contexts
Decision-making and quality trade-off research	Research on architectural decisions, technical debt, prioritization, and socio-technical reasoning	Supports the conceptual framing of software quality as an emergent outcome of cumulative engineering decisions

identify educational emphases and the ways in which decision-making and value-oriented reasoning were represented.

Second, empirical studies addressing skills gaps and industrial expectations were examined to identify how decision-related competencies were discussed in relation to software quality and professional preparedness.

Third, state-of-practice investigations were examined to identify how requirements-related and other quality-relevant decisions were described in professional software engineering contexts.

Across all stages of the synthesis, particular attention was given to distinguishing evidence originating directly from computing education research from evidence derived from industrial and state-of-practice studies. Educational sources were used primarily to characterize curricular framings and competency emphases, whereas industrial sources were used to examine how decision-making and quality-related trade-offs manifest in professional software engineering practice.

The purpose of the synthesis was not to evaluate particular educational interventions or establish causal relationships between education and software quality outcomes, but to construct an interpretative conceptual framework integrating patterns identified across the selected educational and industrial literature.

5. Decision Visibility in Software Engineering Education

This section examines how quality-relevant decision reasoning is represented across software engineering education. Drawing on software engineering and requirements engineering education research, skills-gap studies, and curriculum frameworks, the analysis acknowledges that students already encounter decision-oriented reasoning through activities such as project work, stakeholder interaction, design choices, evidence evaluation, and quality trade-offs. The focus of the synthesis is therefore not whether decision-making is present or absent, but how explicitly choices, assumptions, alternatives, trade-offs, and rationales are made visible across different educational contexts.

5.1. Requirements Engineering Education and Invisible Trade-Offs

Research on requirements engineering education documents a broad range of teaching approaches, including instruction in elicitation, specification, modeling, and validation, as well as project-based, stakeholder-oriented, and other forms of experiential learning Daun *et al.* (2023); Cico *et al.* (2021). These approaches demonstrate that requirements engineering education is not limited to procedural instruction and that students may encounter realistic requirements problems, stakeholder interaction, and contextual complexity during their studies.

At the same time, the presence of authentic or project-based learning activities does not necessarily imply that the decisions embedded in those activities are made explicit objects of instruction and assessment. The distinction relevant to the present study is therefore not between procedural and realistic teaching, but between participating in a decision-rich activity and being explicitly required to identify, justify, compare, and reflect on the quality-relevant decisions that the activity entails.

From this perspective, existing requirements engineering education provides an important foundation for decision-oriented learning while leaving room for a more systematic emphasis on decision visibility. Activities involving elicitation, stakeholder negotiation, requirements evolution, and project work can create opportunities for trade-off reasoning; the model proposed here extends these practices by asking whether prioritization choices, competing stakeholder concerns, assumptions, rejected alternatives, and downstream quality implications are explicitly articulated and assessed.

5.2. Decision-Making Across Professional Competency Frameworks

Studies of skills gaps and competency challenges in software engineering identify problem-solving, critical thinking, and understanding of business or user contexts among the competencies relevant to professional preparedness Akdur (2022); Oguz and Oguz (2019); Diniz *et al.* (2024); de Souza *et al.* (2024); Metrôlho *et al.* (2022). These capabilities are commonly discussed as transversal or professional competencies alongside more narrowly defined technical abilities such as programming or testing.

From the perspective developed here, these competencies and related recommendations for experiential learning and industry involvement create opportunities to examine whether quality-relevant decisions are made explicit for instruction, reflection, and assessment Akdur (2022); Oguz and Oguz (2019).

5.3. Computer Science Curricula 2023: Existing Decision Reasoning and Opportunities for Integration

The *Computer Science Curricula 2023*, jointly developed by ACM, IEEE-CS, and AAAI, presents a contemporary and comprehensive vision of computing education Kumar *et al.* (2024). The curriculum emphasizes foundational knowledge, professional practice, ethical awareness, and sensitivity to social and organizational contexts McKenzie *et al.* (2024). Importantly, decision reasoning is already explicit in parts of the curriculum. Learning

outcomes include comparing software designs in terms of design qualities and evaluating trade-offs among testing approaches, while broader curricular guidance also recognizes constraints, stakeholder concerns, and the impacts of computing decisions.

These elements show that trade-off reasoning and contextual judgment are not absent from contemporary computing curricula. The opportunity addressed by the present model is therefore not to introduce decision-making where none exists, but to integrate such existing elements more systematically through decision visibility. In particular, the model foregrounds whether quality-relevant choices, alternatives, assumptions, and rationales are made explicit and assessable across learning activities, rather than remaining distributed across individual topics or outcomes. Decision visibility is thus proposed as an integrative educational mechanism that extends existing curricular attention to design qualities, testing trade-offs, professional context, and engineering judgment.

5.4. Correctness-Centered Educational Framing and Quality-Oriented Reasoning

Taken together, the reviewed literature shows that software engineering education combines technical and process-oriented instruction with a range of decision-oriented learning opportunities. The prominence and explicitness of decision reasoning nevertheless vary across educational contexts. From the perspective of this paper, the relevant distinction is therefore whether value judgments, prioritization choices, quality trade-offs, assumptions, and alternatives become explicit objects of learning and assessment alongside technical execution.

When educational environments emphasize correctness, compliance, and successful task completion without also examining the reasoning behind quality-relevant choices, students may receive fewer opportunities to practice explicit reasoning about value, prioritization, trade-offs, and acceptable compromise. In terms of the proposed model, this represents lower decision visibility rather than an absence of decision-making education.

6. Industrial Practice and the Visibility of Quality-Relevant Decisions

This section examines industrial software engineering practice as a context in which quality-relevant decisions arise under uncertainty, competing priorities, and real-world constraints. Drawing on state-of-practice investigations in requirements engineering and software development, the analysis considers how prioritization, trade-offs, and contextual judgment are represented in professional practice.

6.1. Quality-Critical Decisions in Requirements and Elicitation

Empirical studies of industrial requirements practice describe elicitation, negotiation, and requirements evolution as involving decisions under incomplete, ambiguous, and changing information Franch *et al.* (2023). Such conditions can require judgments about scope, priorities, stakeholder concerns, and acceptable levels of risk.

Industrial requirements may evolve through interaction among stakeholder needs, technical constraints, and organizational pressures Franch *et al.* (2023). Decisions to implement, defer, or revise requirements can involve trade-offs among functionality, quality attributes, delivery constraints, and available resources, with potential implications for usability, maintainability, and other quality concerns. Such decisions may also be collaborative or informally documented, and decision authority can be distributed across different professional roles.

6.2. *Decision-Making Under Uncertainty and Time Pressure*

Industrial software development can involve uncertainty and time pressure as organizational priorities, stakeholder expectations, and available information change over time Franch *et al.* (2023). Under such conditions, engineering decisions may require professional judgment and contextual reasoning in addition to formal requirements and procedures.

State-of-practice evidence also illustrates situations in which teams accept technical debt, adjust testing effort, or simplify requirements in response to short-term constraints Franch *et al.* (2023). These examples show why quality-relevant decisions may involve explicit or implicit trade-offs whose consequences depend on technical, organizational, and contextual factors. They motivate attention to decision-oriented reasoning without establishing that decision competence alone determines subsequent software quality.

6.3. *Implicit Product Roles of Software Engineers*

Software engineers may contribute to product-related decisions even when formal product ownership resides elsewhere Franch *et al.* (2023). Their technical work can inform requirements interpretation, feasibility judgments, prioritization, and system evolution, thereby connecting engineering choices with stakeholder concerns and longer-term consequences.

The extent of such involvement depends on organizational roles, team structures, decision authority, and organizational context. Product thinking, as defined in this paper, therefore concerns the reasoning involved in quality-relevant engineering choices rather than assuming that software engineers hold formal product responsibility or sufficient authority to implement particular decisions.

6.4. *Decision Visibility in Industrial Software Engineering Practice*

Taken together, the industrial evidence considered here illustrates professional contexts in which requirements, priorities, constraints, and quality concerns may evolve and require continuing judgment. Decisions made during requirements elicitation, prioritization, design, testing, and system evolution can therefore provide relevant contexts for examining trade-offs, uncertainty, and decision rationale.

The educational significance of this evidence is conceptual rather than causal. Compared with learning activities in which requirements, constraints, and expected outcomes are more fully specified, these industrial contexts illustrate decision conditions that can be

represented more explicitly in educational activities. The proposed model uses this contrast to motivate decision visibility without assuming that educational differences cause particular professional behaviors or software quality outcomes.

7. Interpreting Software Quality Through Decision Visibility and Product-Thinking Practices

This section interprets the evidence synthesized in the previous sections through the conceptual model proposed in this paper. The discussion considers how educational framing and decision visibility may relate to opportunities to practice quality-oriented reasoning about prioritization, trade-offs, uncertainty, and engineering responsibility.

7.1. Educational Framing and the Visibility of Engineering Responsibility

The literature reviewed in Section 5 shows that software engineering education includes both technically focused instruction and forms of authentic, stakeholder-oriented, and decision-oriented learning. Within this varied landscape, the present model focuses specifically on whether the rationale behind quality-relevant choices is made visible for discussion and assessment. When learning activities emphasize artifact correctness or process compliance without also requiring students to explain trade-offs, assumptions, and prioritization, opportunities to practice explicit quality-oriented reasoning may be reduced. Decision visibility therefore provides a way to examine how educational activities make engineering responsibility available for reflection and assessment, without assuming that such reasoning is generally absent from software engineering education.

7.2. Decision Visibility as a Proposed Link Between Education and Quality-Oriented Reasoning

The industrial evidence discussed in Section 6 illustrates professional contexts in which engineers encounter evolving requirements, constraints, prioritization, and quality-related trade-offs. As discussed in Section 5, software engineering education already includes forms of trade-off reasoning, authentic project work, stakeholder engagement, and decision-oriented learning. The issue addressed by the present model is therefore not the absence of such practices, but how consistently quality-relevant decisions and their rationales are made visible across learning activities.

Decision visibility is proposed as a link between educational framing and opportunities to develop product-thinking practices and quality-oriented reasoning. Making choices, alternatives, assumptions, and trade-offs explicit creates opportunities for students to articulate and examine the reasoning behind engineering decisions. This proposed relationship should not be interpreted as evidence that decision visibility causes professional competence or improved software quality outcomes.

Observable indicators for future empirical study include the ability to articulate decision rationale, justify quality-related trade-offs, explain prioritization under uncertainty,

and connect technical decisions to relevant quality consequences. These indicators frame product thinking as a form of engineering reasoning that can be practiced and assessed without assuming that the proposed educational relationships have already been empirically demonstrated.

7.3. Relating Decision Visibility to Software Quality Reasoning

The proposed model complements technical and process-oriented approaches to software quality by focusing on the reasoning that accompanies engineering choices. Its educational implication is limited to making decision rationale, relevant assumptions, trade-offs, and anticipated quality consequences explicit objects of discussion and assessment. Whether such practices subsequently affect professional behavior or software quality outcomes remains an empirical question.

8. Design Principles and Educational Applications

The conceptual model proposed in this paper suggests that software engineering education can make quality-relevant decisions more explicit during learning activities. This section translates the conceptual argument into a set of educational design principles and classroom-oriented applications intended to support the development of product-thinking practices and quality-oriented reasoning.

8.1. Educational Design Principles

The proposed model suggests that decision-oriented reasoning can be integrated into existing software engineering education without reducing technical rigor or requiring a separate curricular framework. The design principles below complement technical instruction by making choices, rationale, uncertainty, prioritization, and quality-related trade-offs more explicit in learning and assessment.

Based on the conceptual model developed in this paper, five educational design principles can be proposed for software engineering education.

DP1: Make requirements ambiguity intentional. Assignment instructions should clearly specify the learning task, expected deliverables, and assessment criteria, while the product problem itself may intentionally contain incomplete, evolving, or conflicting requirements. The ambiguity should therefore reside in the engineering situation that students must reason about, not in what students are expected to do for the assignment.

DP2: Assess decision rationale in addition to technical correctness. Students should be evaluated not only on the correctness of artifacts, but also on their ability to justify alternatives, explain trade-offs, and articulate quality-related consequences.

DP3: Frame quality attributes as potentially competing commitments. Learning activities should require students to identify and justify context-dependent trade-offs among relevant quality attributes, such as maintainability, performance, usability, reliability, and security, while also considering constraints such as delivery time, budget, and

Table 5
Example classroom-oriented applications of decision-oriented product thinking

Educational activity	Task structure	Decision-oriented competence emphasized
Requirements priority memo	Students receive multiple competing stakeholder requests with limited implementation capacity and must justify prioritization choices, rejected features, and anticipated quality consequences.	Stakeholder reasoning, prioritization, and quality trade-off awareness.
Testing trade-off brief	Students choose among competing testing activities under limited time and explain how risk, maintainability, usability, and performance concerns influenced their decisions.	Risk-oriented quality reasoning and explicit trade-off articulation.
Technical debt decision log	Students implement a feature under delivery pressure and later document accepted shortcuts, associated risks, and long-term quality implications.	Lifecycle reasoning and accountability for future quality consequences.
Architecture choice comparison	Students compare alternative architectures and justify decisions in terms of maintainability, scalability, user impact, and future evolution.	Architecture-quality reasoning and explicit justification of trade-offs.
Sprint reflection exercise	Students identify a decision that most affected software quality during a project iteration and explain which stakeholder priorities and quality attributes shaped the outcome.	Reflection on decision visibility and quality-oriented reasoning.

available resources. The aim is not to assume that quality attributes necessarily conflict, but to make students examine when competing priorities or constraints require an explicit choice.

DP4: Use technical debt as a product-thinking exercise. Students should explicitly reflect on how short-term implementation shortcuts influence long-term maintainability, integration risk, and software quality.

DP5: Make considered alternatives and rejection rationales visible. As one way of providing the decision rationale required by DP2, students can identify relevant alternatives they considered and explain why particular options were rejected. This makes the decision boundary and its underlying trade-offs available for reflection and assessment without requiring an exhaustive record of every possible option.

8.2. Classroom-Oriented Applications

The proposed conceptual model can be translated into concrete software engineering learning activities that make quality-relevant decisions more visible and assessable during education. Table 5 summarizes several example applications aligned with decision-oriented product thinking.

For example, the *testing trade-off brief* in Table 5 can be assessed using the indicators in Table 3. Under a fixed release window, a student might prioritize security and reliability testing over additional performance testing, identify the affected stakeholders, state relevant assumptions, and explain a rejected alternative. A subsequent change in a stakeholder requirement can then require the student to revise or retain the original priority and justify that response. Assessment focuses on the evidence used, the explicit trade-off, and the rationale for revising or retaining the decision rather than on agreement with a pre-determined "correct" testing strategy. Where AI-generated code is used, students remain responsible for explaining how they checked it and for justifying the resulting engineering decisions; generated code itself is not evidence of decision rationale.

8.3. *Implications for Industry-Academia Collaboration*

The conceptual model proposed in this paper suggests that industry-academia collaboration can focus not only on technological alignment, but also on exposing students to realistic decision conditions. Internships, capstone projects, and collaborative industry assignments may provide opportunities for students to engage with ambiguity, prioritization pressures, competing stakeholder expectations, and long-term quality consequences in ways that increase decision visibility during learning Akdur (2022); Oguz and Oguz (2019).

8.4. *Implications for Future Research*

The conceptual model proposed in this paper identifies several questions for future research on decision-oriented reasoning in software engineering education. In particular, empirical studies could examine how students recognize, justify, and communicate quality-relevant decisions across requirements engineering, architecture, testing, maintenance, and other software engineering learning contexts.

Future research could examine how decision-oriented reasoning develops over time, how it interacts with organizational and project contexts, and whether and under what conditions it relates to professional software quality practices or outcomes. Empirical studies of rationale articulation, trade-off reasoning, and value assessment could test the proposed relationships among educational framing, decision visibility, product-thinking practices, and quality-oriented reasoning.

Furthermore, the concept of product thinking articulated in this paper provides a potential lens for integrating socio-technical and technical perspectives in future software engineering research.

Taken together, these design principles and educational applications operationalize the conceptual model proposed in this paper by translating decision visibility into concrete pedagogical practices. Rather than treating product thinking as a separate managerial domain, the framework positions it as a teachable form of engineering reasoning grounded in explicit engagement with trade-offs, uncertainty, prioritization, and software quality consequences.

9. **Boundary Conditions, Rival Explanations, and Limitations**

As a conceptual and integrative study, the framework proposed in this paper is subject to several boundary conditions and interpretative limitations. The paper does not claim direct causal relationships between software engineering education and software quality outcomes. Rather, it proposes decision visibility as one plausible conceptual mechanism linking educational framing with quality-oriented reasoning relevant to professional software engineering practice.

First, the framework developed in this study is derived from interpretative synthesis rather than from primary empirical investigation. Although the analysis draws on system-

atic reviews, mapping studies, and industrial state-of-practice research, alternative interpretations of the same evidence remain possible. The proposed model should therefore be understood as a theoretically grounded explanatory perspective rather than as an empirically validated causal account.

Second, software quality outcomes are shaped by multiple interacting factors beyond educational framing alone. Organizational culture, process maturity, management incentives, team communication practices, market pressure, governance structures, tooling ecosystems, and technical debt management practices may all substantially influence how software-quality decisions are made in practice.

Accordingly, the present study does not argue that educational framing alone determines software quality outcomes. Instead, it positions education as one plausible contributor to the development of quality-oriented reasoning within broader socio-technical and organizational environments.

Third, the relevance of the proposed framework may vary across educational contexts. Decision visibility may already be higher in studio-based, agile, project-based, capstone-oriented, or industry-partnered learning environments where trade-offs, ambiguity, and stakeholder negotiation are explicit parts of educational practice. The framework is therefore intended to support analysis of variation in decision visibility rather than to characterize software engineering curricula uniformly.

Finally, decision-oriented product thinking remains a conceptual construct that requires additional empirical operationalization and validation. Although the framework proposed in this study identifies observable indicators related to trade-off reasoning, uncertainty handling, prioritization, and rationale articulation, further research is needed to examine how such forms of reasoning develop across educational and professional contexts.

Despite these limitations, the framework developed in this paper provides a conceptual lens for examining proposed relationships between the educational visibility of software-quality decisions and quality-oriented reasoning. By integrating educational framing, decision visibility, and product-thinking practices, the study provides a foundation for future empirical and educational research on these proposed relationships.

10. Conclusion

Persistent software quality problems continue to motivate research on technical, process, organizational, and educational aspects of software engineering. This paper has developed a conceptual model for examining how educational framing and decision visibility may relate to opportunities to practice product-thinking and quality-oriented reasoning.

By integrating evidence from software engineering education research, industry-oriented skills-gap studies, and state-of-practice investigations in requirements engineering, the paper identified an opportunity to connect existing forms of decision-oriented learning more systematically. Contemporary software engineering education already includes contexts in which students encounter design choices, stakeholder concerns,

evidence-based decisions, and quality trade-offs; the conceptual issue examined here is how such quality-relevant decisions and their rationales can be made more consistently visible across learning activities.

The central contribution of this work lies in conceptualizing decision visibility as a proposed link between educational framing and opportunities to practice quality-oriented reasoning. From this perspective, software quality education can address not only verification activities, but also how students recognize, evaluate, justify, and communicate quality-relevant engineering decisions.

By positioning product thinking as a decision-oriented engineering competence rather than as a managerial role, the framework provides a basis for examining quality-oriented reasoning under conditions of ambiguity, competing constraints, and uncertainty. The study further translates this framework into educational design principles and classroom-oriented applications intended to make trade-offs, rationale articulation, and engineering responsibility more visible during software engineering education.

The educational implication of the proposed model is that quality-relevant decisions can be made explicit objects of learning, reflection, and assessment alongside technical methods and processes. In this sense, product thinking is positioned not as an external business concern, but as a teachable form of engineering reasoning for examining value, uncertainty, trade-offs, and software quality consequences.

Future research may empirically examine how decision visibility, rationale articulation, and trade-off reasoning develop across different software engineering educational environments and how these forms of reasoning relate to professional software quality practices over time.

Funding

This research received no external funding.

Declarations

Conflict of Interest.

The author declares no conflict of interest.

Ethical Approval.

This study is based on a conceptual and integrative analysis of previously published literature and does not involve human participants, personal data, or experimental procedures requiring ethical approval.

Use of Artificial Intelligence Tools.

Artificial intelligence-based tools were used solely for stylistic and grammatical refinement, including language correction and paraphrasing for clarity. All scientific content,

interpretations, analyses, and conclusions are the author's original work, and the author takes full responsibility for the manuscript.

References

- Akdur, D. (2022). Analysis of software engineering skills gap in the industry. *ACM Transactions on Computing Education*, 23(1), 1–28.
- Bugayenko, Y., Bakare, A., Cheverda, A., Farina, M., Kruglov, A., Plaksin, Y., Pedrycz, W., Succi, G. (2023). Prioritizing tasks in software development: A systematic literature review. *Plos one*, 18(4), 0283838.
- Bukhsh, F.A., Bukhsh, Z.A., Daneva, M. (2020). A systematic literature review on requirement prioritization techniques and their empirical evaluation. *Computer Standards & Interfaces*, 69, 103389.
- Cámara, J., Wohlrab, R., Garlan, D., Schmerl, B. (2023a). ExTrA: Explaining architectural design tradeoff spaces via dimensionality reduction. *Journal of Systems and Software*, 198, 111578.
- Cámara, J., Wohlrab, R., Garlan, D., Schmerl, B. (2023b). Focusing on what matters: Explaining quality tradeoffs in software-intensive systems via dimensionality reduction. *IEEE Software*, 41(1), 64–73.
- Cico, O., Jaccheri, L., Nguyen-Duc, A., Zhang, H. (2021). Exploring the intersection between software industry and Software Engineering education-A systematic mapping of Software Engineering Trends. *Journal of Systems and Software*, 172, 110736.
- Daun, M., Grubb, A.M., Stenkova, V., Tenbergen, B. (2023). A systematic literature review of requirements engineering education. *Requirements Engineering*, 28(2), 145–175.
- de Souza, A.S., de Oliveira, J.L., da Costa Paiva, S.L., de Vasconcelos, A.M.L. (2024). Software Engineering Competency Challenges. In: *Workshop sobre Aspectos Sociais, Humanos e Econômicos de Software (WASHES)*, pp. 181–186. SBC.
- de Toledo, S.S., Martini, A., Sjøberg, D.I. (2021). Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study. *Journal of Systems and Software*, 177, 110968.
- Diniz, W., Valença, M., França, C., Santos, A., Pincovsky, M. (2024). The skill gap in software industry: A mapping study. In: *Simpósio Brasileiro de Engenharia de Software (SBES)*, pp. 192–200. SBC.
- Dobslaw, F., Angelin, K., Öberg, L.-M., Ahmad, A. (2023). The gap between higher education and the software industry—A case study on technology differences. In: *Proceedings of the 5th European Conference on Software Engineering Education*, pp. 11–21.
- Fernández, D.M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., Conte, T., Christiansson, M.-T., Greer, D., Lassenius, C., et al. (2017). Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice. *Empirical software engineering*, 22(5), 2298–2338.
- Franch, X., Glinz, M., Mendez, D., Seyff, N. (2021). A study about the knowledge and use of requirements engineering standards in industry. *IEEE Transactions on Software Engineering*, 48(9), 3310–3325.
- Franch, X., Palomares, C., Quer, C., Chatzipetrou, P., Gorschek, T. (2023). The state-of-practice in requirements specification: an extended interview study at 12 companies. *Requirements Engineering*, 28(3), 377–409.
- Garousi, V., Giray, G., Tuzun, E., Catal, C., Felderer, M. (2019). Closing the gap between software engineering education and industrial needs. *IEEE software*, 37(2), 68–77.
- Habiba, U.-e., Haug, M., Bogner, J., Wagner, S. (2024). How mature is requirements engineering for AI-based systems? A systematic mapping study on practices, challenges, and future research directions. *Requirements Engineering*, 29(4), 567–600.
- Jorgensen, M., Dyba, T., Kitchenham, B. (2005). Teaching evidence-based software engineering to university students. In: *11th IEEE International Software Metrics Symposium (METRICS'05)*, p. 8. IEEE.
- Junior, H.J., Travassos, G.H. (2022). Consolidating a common perspective on Technical Debt and its Management through a Tertiary Study. *Information and Software Technology*, 149, 106964.
- Kotti, Z., Galanopoulou, R., Spinellis, D. (2023). Machine learning for software engineering: A tertiary study. *ACM Computing Surveys*, 55(12), 1–39.
- Kumar, A.N., Raj, R.K., Aly, S.G., Anderson, M.D., Becker, B.A., Blumenthal, R.L., Eaton, E., Epstein, S.L., Goldweber, M., Jalote, P., et al. (2024). Computer science curricula 2023. ACM.
- Lenberg, P., Feldt, R., Wallgren, L.G. (2015). Behavioral software engineering: A definition and systematic literature review. *Journal of Systems and software*, 107, 15–37.
- Marijan, D., Gotlieb, A. (2021). Industry-Academia research collaboration in software engineering: The Certus model. *Information and software technology*, 132, 106473.

- Mckenzie, S., Fernando, N., Dias, I., Cheng, B., Hoang, T., Liu, X. (2024). Integrating Human-Centric Approaches into Undergraduate Software Engineering Education: A Scoping Review and Curriculum Analysis in the Australian Context. In: *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops*, pp. 100–107.
- Metrólho, J., Ribeiro, F., Graça, P., Mourato, A., Figueiredo, D., Vilarinho, H. (2022). Aligning software engineering teaching strategies and practices with industrial needs. *Computation*, 10(8), 129.
- Mittas, N., Trygoniari, D., Ampatzoglou, A., Arvanitou, E.-M., Volioti, C., Chatzigeorgiou, A., Angelis, L. (2025). Required Knowledge, Skills and Transversal Competences for a Career in Software Engineering. *Information and Software Technology*, 107949.
- Nocera, S., Romano, S., Francese, R., Scanniello, G. (2025). Software engineering education: Results from a training intervention based on SonarCloud when developing web apps. *Journal of Systems and Software*, 222, 112308.
- Oguz, D., Oguz, K. (2019). Perspectives on the gap between the software industry and the software engineering education. *Ieee Access*, 7, 117527–117543.
- Öqvist, K., Messinger, J., Wohlrab, R. (2024). Supporting Early Architectural Decision-Making through Trade-off Analysis: A Study with Volvo Cars. In: *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pp. 411–416.
- Pan, R., Bagherzadeh, M., Ghaleb, T.A., Briand, L. (2022). Test case selection and prioritization using machine learning: a systematic literature review. *Empirical Software Engineering*, 27(2), 29.
- Ruhe, G. (2002). Software engineering decision support—a new paradigm for learning software organizations. In: *International Workshop on Learning Software Organizations*, pp. 104–113. Springer.
- Song, Q., Runeson, P. (2023). Industry-academia collaboration for realism in software engineering research: Insights and recommendations. *Information and Software Technology*, 156, 107135.
- Tang, A., Kazman, R. (2021). Decision-making principles for better software design decisions. *IEEE Software*, 38(6), 98–102.
- Villoth, J.P., Zivkovic, M., Zivkovic, T., Abdel-salam, M., Hammad, M., Jovanovic, L., Simic, V., Bacanin, N. (2025). Two-tier deep and machine learning approach optimized by adaptive multi-population firefly algorithm for software defects prediction. *Neurocomputing*, 630, 129695.
- Wohlin, C., Runeson, P. (2021). Guiding the selection of research methodology in industry–academia collaboration in software engineering. *Information and software technology*, 140, 106678.
- Živković, T., Živković, M. (2020). Comparative analysis of techniques for testing object oriented programs. In: *2020 Zooming Innovation in Consumer Technologies Conference (ZINC)*, pp. 270–275. IEEE.
- Zivkovic, T., Zivkovic, M. (2021). Survey of learning environments for software testing education. In: *7th Conference on the Engineering of Computer Based Systems*, pp. 1–9.
- Zivkovic, T., Draskovic, D., Nikolic, B. (2023a). Learning environments in software testing education: An overview. *Computer Applications in Engineering Education*, 31(6), 1497–1521.
- Zivkovic, T., Nikolic, B., Simic, V., Pamucar, D., Bacanin, N. (2023b). Software defects prediction by meta-heuristics tuned extreme gradient boosting and analysis based on shapley additive explanations. *Applied Soft Computing*, 146, 110659.