# Building a Decision Landscape Model for Software Development: From Empirical Insights to Formal Theory

Hannes Salin[a]
*School of Information and Engineering, Dalarna University, Borlänge, Sweden*

Keywords: Decision-Making, Value Stream Mapping, Formal Model, Software Engineering, Software Development.

Abstract: Value stream mapping is a well known method for identifying waste and bottlenecks in production processes. It is also used in the software engineering context, to map value streams of different processes in software development. By regarding decision-making processes as value streams, we can describe and further analyze decision-making flows in an organization for optimization such as reduced lead times or decision making efficiency. Using empirical data from three different software development organizations in Sweden, we develop a formal model to describe decision-making flows in a software development organizational context, in a compact and systematic syntax. Our formal model can thus be used for analyzing decision-making flows and support management in better understanding how decisions are made within their organizations.

## 1 INTRODUCTION

As for any organization in software engineering that produces value of some sort, decision-making is crucial, e.g. for improved managerial perspectives with metrics (Salin et al., 2022), architectural design (Razavian et al., 2019), or product development (Mendes et al., 2018). In an organization with many different collaborative teams, units and processes, it may be complicated to fully understand the overall *decision-making landscape*, namely the mapping of all necessary decision flows that needs to be executed for efficient or optimal productivity. For this reason, it would be beneficial to identify and map decision processes and associated bottlenecks in the organization, in order to better understand how these flows can be optimized. Also, it might be the case that formal decision-making processes are described in documentation and policies, but works differently in practice. For that reason, organizations need to map such current processes to better understand their own workflows.

Value stream mapping (VSM) is a powerful tool in the lean (and agile) toolbox, which also has been shown to be suitable for managerial analysis (Eleftherios Andreadis and Kumar, 2017). VSM is a systematic approach used to visualize and analyze the flow of a process, identifying inefficiencies, bottlenecks, and potential improvements. Such mapping supports decision-making for prioritizing and coordinating continuous improvement initiatives, by mapping each step of the workflow and thereby distinguishing value-added from non-value-added activities. The typical method to perform a VSM is via workshops (Mäkinen, 2024), although different qualitative and/or quantitative methods and other type of documenting activities works as well. VSM is predominantly used in manufacturing and operations contexts, however, also used in software engineering and DevOps contexts (Botero et al., 2024; Mäkinen, 2024; Murphy and Kersten, 2020; Tankhiwale and Saraf, 2020). The term *value* is broad and can include anything that the organization regards (and define) as valuable, either in time, monetary units or other type of metrics. The value of a decision outcome would be one example.

Given a compact and understandable formal model for decision-making processes in any context of software engineering, we aim to provide a supportive tool to perform a VSM directed towards decision value flows, easy to analyze for optimizations. Moreover, by introducing a formal model as basis for running a VSM, it could potentially be analyzed with more depth compared to standard approaches using multiple step workshops or flow-chart based processes with natural language descriptions (Qin and Liu, 2022; Meudt et al., 2017). Since VSM is not necessarily only suited for one context or domain, any

[a] https://orcid.org/0000-0001-6327-3565

Salin, H.
Building a Decision Landscape Model for Software Development: From Empirical Insights to Formal Theory.
DOI: 10.5220/0013457300003964
In *Proceedings of the 20th International Conference on Software Technologies (ICSOFT 2025)*, pages 15-26
ISBN: 978-989-758-757-3; ISSN: 2184-2833

improvements that support such approach would be general in nature.

## 1.1 Paper Outline

The rest of the paper is outlined as follows: the remaining of this section includes related work and formulated research question. Section 2 describe the chosen methodology and theoretical foundation, whereas section 3 presents the results from the empirical data collection. Section 4 describe the developed model, and section 5 the discussion including threats to validity and future work. Section 6 concludes our research.

## 1.2 Related Work

We underscore that our research is not within the field of traditional decision theory (Kumar, 2025), nor the cognitive sciences in decision-making (Wallsten, 2024). Instead, our work is in particular focused on the ability to transform observed decision-making processes in software engineering organizations into compact and comprehensible formal language. In contrast, much research is made in general methods for logic of decision-making, e.g. a general model for decisions under incomplete knowledge (Marković et al., 2024), or different decision-making methods for artificial intelligence and systems such as prompting and reinforcement learning (Yang et al., 2023).

Closest to our approach is the Managerial Decision-making Description Model (MDDM) that has been proposed, which can be used for visualization of decision-making processes that changes the business structure states (Kunigami et al., 2019). Focus for MDDM is on graphical representations via diagrams connecting agents' goals, events, objectives and other business components. However, the model does not provide a compact mathematical syntax for describing the decision-making landscape.

Regarding supportive methods to VSM for decision-making mapping, discrete events simulation has been proposed with VSM towards investment decisions (Helleno et al., 2015), and another type of simulation to enhance lean optimization calculations (Liu and Yang, 2020). One study implemented a multi-agent system for dynamic VSM for manufacturing systems, where the proposed method consisted of a cyber-physical system with hardware components and sensors (Huang et al., 2019). However, none of these approaches has utilized a formal syntax model for describing decision-making processes.

Finally, a strict formal approach is the Burrows-Abadi-Needham (BAN) logic for analyzing and reason about authentication in security protocols (Burrows et al., 1990). However, decision-making processes are not in scope for BAN logic, although our proposed model uses a similar formalized syntax with defined relations in order to systematically describe complex processes. Thus, to our knowledge, there are no proposed formal models to describe decision-making flows using a more syntax-based approach, with the aim to support VSM.

## 1.3 Research Question

Given the importance of understanding a software development organization's current decision landscape, we need a model where decision boundaries, unclear mandates and other challenges can be identified. Also, to better optimize the organization, a decision landscape map is required, thus a model for how to efficiently construct such mapping is needed. A formal model is a mathematical or logical representation of a system or phenomenon, defined by precise rules and structures to analyze, predict, and/or verify its behavior. For that reason, we formulate the following research question:

- *RQ1: How can a formal model of decision-making in software development organizations be used for supporting value stream mapping?*

By constructing a formal model, our study aims to provide a compact way to describe decision-making processes as a supporting tool for VSM, hence ensuring practical applicability for studying software development organizational decision-making. Thus, we seek a way for engineering managers to better understand and optimize their organizations using formal language to describe current decision-making flows.

## 2 METHOD

The chosen research methodology follows an inductive approach aimed at theory building from empirical data (Sjøberg et al., 2008), collected in several instances of the software development context (different roles from three Swedish software engineering companies). The approach is exploratory in the sense of identifying current decision-making processes, which then render a theoretic model construction. Data was gathered through semi-structured interviews and an open question survey to capture decision-making processes, potential conflicts, and dependencies across different organizational teams, roles and units. Target companies were chosen via convenience sampling. An iterative process of qualitative pattern identification, using thematic-inspired analysis, was used to summarized a set of key insights. These insights were then used to develop the proposed formal model of decision-making.

The aim is to develop a *descriptive* model since its primary goal is to provide a compact but structured language to represent and analyze observed decision-making processes (Wang et al., 2004). It should capture relationships and to some degree the dynamics of decisions without prescribing how these processes should ideally function or be optimized. Therefore, unlike normative models, it does not aim to define optimal decision-making practices but instead serves as a tool for understanding and documenting real-world workflows in conjunction with VSM practices.

## 2.1 Theoretical Foundation

Based on the guidelines in theory building for software engineering (Sjøberg et al., 2008), a theory should be presented with certain archetypes. We have chosen to use two of the defined concepts: *actors* and *activities*, where actors are different roles in an organization and activities refers to different decision-making relations, e.g. making a successful/unsuccessful decision and so on. Moreover, we use the notion of *constructs, propositions, explanations* and *scope* to construct and present the proposed model. Constructs are explicit classes or attributes, e.g. a design, a concept or an organizational entity. A proposition is a statement aligned with one or many constructs, e.g. *"decisions in a team are not violated by a first line manager"*. Explanations are descriptions of why the propositions are formulated, and scope details the motivation for the whole theory.

We construct the decision-making model on the basis of *bounded rationality* (Simon, 1955) for how agents behave, and extends this foundation by incorporate Hackman's unit authority theory (Hackman, 1986). By combining bounded rationality as the underlying principle with inspiration from Hackman's authority classifications, our model captures the nuances of hierarchical and collaborative decision-making.

### 2.1.1 Bounded Rationality

The notion of *bounded rationality* is the assumption that individuals and teams make decisions within the limits of their knowledge, resources, and cognitive capacities (Simon, 1955). We use bounded rationality as a foundational principle in our model, recognizing that decision-makers in software development organizations cannot evaluate all possible options exhaustively due to time constraints, limited information, and complex inter-dependencies. Instead of seeking optimal solutions, we assume the decision-maker instead aims for satisfactory outcomes by using heuristics. This assumption aligns with the realities of agile and dynamic environments in software engineering, where exhaustive decision-making is impractical (Erbas and Erbas, 2009; Moe et al., 2012). Although bounded rationality and similar rational decision models are the commonly used assumptions used in software engineering (Ralph, 2018), other standpoints exists as well such as empirical naturalistic theory (Pretorius et al., 2024). However, this is more studied towards software design decision-making context, whereas our study focus on the software development organizational context.

### 2.1.2 Hackman's Theory

Hackman's unit authority theory (Hackman, 1986) classifies decision-making authority within teams into three levels: managerial control (decisions are centralized with managers), shared control (authority is distributed between managers and team members) and self-control (teams operate autonomously with full decision-making power). This classification provides a conceptual foundation for understanding how authority impacts team dynamics and decision-making processes. The theory has been used previously in the agile software engineering context to study team autonomy (Moe et al., 2021).

## 3 RESULTS

In this section we describe the resulting data collection procedure, along with the data. We summarize the findings in the data into a set of key insights, which are later used for motivating the model. These key insights describes identified themes from the data, i.e. expressions and descriptions from the participants that align within certain themes.

## 3.1 Data Collection

The data collection included three Swedish organizations with software development as part of their operations: two in banking/financial institutions (referred to as company A and company B, respectively) and one public sector agency. These companies were selected by convenience sampling, and all three had similar product based organizations with agile software development.

Semi-structured interviews and one online survey was created for collecting data about current decision-making processes. The survey was open-ended with a free text answer field, where participants was asked to describe what roles that made what type of decisions, and if there were any necessary decisions that could not be made for a certain role, and how these were handled in the organization. The survey was sent to $N = 61$ employees in total after a convenience sampling across the three companies. The sampling stemmed from initial discussions with representatives from each company, which directed/recommended subsets of employees to include in the data collection phase. Four different roles were represented (software developers, architects, project managers, engineering managers) with $n = 34$ respondents in total. The survey questions are detailed in Tab. 1 and was further elaborated in the interviews, i.e. the interview protocol was not different from the survey questions. The survey had an introductory text highlighting that all questions were related to decision-making processes in the organization and not towards cognitive aspects; one example was also given to underscore this.

Four interviews were conducted, with two software developers (one from the public sector agency and one from bank/finance company B), one architect (bank/finance company A) and one engineering manager (bank/finance company B). All interviews were remote and scheduled for 20 minutes each. All interviewees were also respondents to the survey.

Table 1: Questions stated in the survey, and used for elaboration in the semi-structured interviews.

| ID | Question |
|---|---|
| $\mathsf{Q}_1$ | Can you describe how decisions are made in your role, from the need of a decision to when it is executed? |
| $\mathsf{Q}_2$ | Can you describe what decisions you need but are not within your mandate? |
| $\mathsf{Q}_3$ | Can you describe what decisions you may need to take for someone else that does not have the same mandate? |
| $\mathsf{Q}_4$ | Can you describe what challenges you face during decision-making processes in your organization? |
| $\mathsf{Q}_5$ | If you cannot take a decision immediately, can you describe the steps you need to take in the organization in order to make the decision? |

From the respondents of the survey, $n_1 = 17$ were from the public sector agency, $n_2 = 9$ from company A, and the remaining $n_3 = 8$ from company B. We did not collect data on the respondents' gender, age or other personal data. The validity of the sample cannot be used for generalization, but rather indicative.

## 3.2 Data Summary

From the survey results we collected and grouped different clusters of similar themes. From the clustering we then formulated different insights, representing each cluster. The discovered clusters were labeled: *same role - different authority, no decision, lack of information, need for escalation, decision dependency*; and denoted $\mathsf{KI}_1 - \mathsf{KI}_5$ respectively. We continued to fill these clusters with data from the interviews. One additional cluster, denoted $\mathsf{KI}_6$, was discovered from the interviews: *enforced decisions*. We summarize the clusters into key insights that were discovered during the data collection phase in Tab. 2. We acknowledge that these insights have their limitations due to a small-size sample, and further data collection would be helpful to refine the development of future versions of the model.

Table 2: Key insights extracted from the data collection phase.

| ID | Key insight |
|---|---|
| $\mathsf{KI}_1$ | One context (team, role) can contain several employees with the same appointed role, but with different decision mandate on the same decision type. |
| $\mathsf{KI}_2$ | Employees with same roles and same mandate might stall in a decision-making process. The same can happen even if one role has higher mandate than the other, and no decision is made. |
| $\mathsf{KI}_3$ | In several cases, a decision could be stalled due to lack of information or knowledge of the decision-maker. The typical implication was that an investigation and/or meetings were scheduled to analyze the decision further. |
| $\mathsf{KI}_4$ | Similar to $\mathsf{KI}_2$ and $\mathsf{KI}_3$, in several occasions a stalled decision could not proceed even though more information was gathered. In these cases, the typical scenario was escalation to management and/or architects for final decision-making. |
| $\mathsf{KI}_5$ | In several cases, a decision-maker was hesitant to make a decision before another dependent decision was made. This could happen both within a team or between different organizational units. |
| $\mathsf{KI}_6$ | Decisions can be enforced, by actors that lack the sufficient mandate. Decisions can also be overridden by actors with- and without sufficient mandates. |

## 4 MODEL DEVELOPMENT

This section describes the proposed formal model for describing decision-making processes in a software development context. We use standard set theory to describe constructs, and relations using the notation $a \oplus b$ where $a, b$ are any constructs and $\oplus$ any defined relation operator within the model. A construct is any set or element of a set defined in model. The basis for our chosen notation is inspired by the work of (Wang et al., 2004).

### 4.1 Decision-Making Context Boundary

Let $C$ represent a specific context, such as a development team, architecture role, IT infrastructure team or a managerial context. The *Decision-Making Context Boundary (DCB)* for context $C$, denoted as $\mathsf{DCB}(C)$, is the set of all decisions that can be made by the actors operating within $C$, subject to the constraints and scope of $C$. To avoid vagueness of what a context is, we formally define it using actors and mandate sets.

**Definition 1** (Actor Set)**.** *Let $a_i$ be an actor, i.e. a role within an organization with specified task assignments. For the contexts in this model we use actors such as* system developer, architect, agile actor, managerial actor. *The set of all $a_i$ that belongs to a specific decision-making context is denoted $A$ and called* actor set.

**Definition 2** (Mandate Set)**.** *Let a tuple $(a_i, (d_j, m_j))$ be a mandate description, which specify the level of mandate $m_j \in \mathbb{N}$ actor $a_i$ has for decision $d_j$. The positive integer $m_j$ thus represents the hierarchical authority level of actor $a_i$ where higher numbers indicate greater decision-making power. For $(a_i, (d, m_j))$ and $(a_k, (d, m_j^*))$ where $m_j > m_j^*$ then $a_i$ has a higher authority level than $a_k$ and can override $a_k$ in decision $d$. The set of all tuples that belongs to a specific decison-making context is denoted $M$ and called the* mandate set.

**Definition 3** (Context)**.** *Let $C$ be a context within a software engineering organization. A context $C$ is defined as a tuple $C = (A, M)$ where $A$ is the* actor set *(or roles), and $M$ the* mandate set *given to the actors.*

From the underlying theoretical assumptions of bounded rationality and Hackman's theory, we are able to define three constraints for an agent when making a decision: *authority, knowledge* and *resource* constraints. These are necessary to be managed by the decision-making agent in order to make a decision:

**Definition 4** (Context Decisions)**.** *We call $d \in D$ a decision, where $D$ is the universal decision space of all possible decisions in the organization. The element $\perp \in D$ represents the absence of a decision, signifying that no explicit action or choice has been made within the context. For a given decision $d$ that is valid to execute in context $C$, we denote it as $d \rightleftharpoons C$, unless there are no constraints on $d$. The following constraints are defined:*

- *Authority Constraint: $d$ can only belong to* $\mathsf{DCB}(C)$ *if $d$ is within the mandate defined in $M$ for an actor in $A$.*
- *Knowledge Constraint: $d$ can only belong to* $\mathsf{DCB}(C)$ *if the necessary information for $d$ is accessible to the actors in $A$, i.e. information that enables an informed decision instead of just make a decision by chance.*
- *Resource Constraint: $d$ can only belong to* $\mathsf{DCB}(C)$ *if the resources required for $d$ are available within the scope of $C$, i.e. people, organizations or assets to be included in the decision.*

*If $d$ is constrained by any of the three constraint types, we denote that by $d \rightharpoonup C$, meaning that this decision attempt will not be valid in $C$ (unsuccessful).*

We now define the $\mathsf{DCB}(C)$ as the set of all decisions $d$ for which $d \rightleftharpoons C$, satisfying the context constraints:

**Definition 5** (Decision-Making Context Boundary)**.**

$$\mathsf{DCB}(C) = \{d \in D \mid d \rightleftharpoons C\} \tag{1}$$

*where $D$ is the universal set of all possible decisions in the system or organization, $d$ an individual decision and $C$ a specific context, such as a team, organizational unit, or domain.*

For instance, let $C$ be an unrealistic (for illustrative purposes) context of a developer team that only has the ability to prioritize a backlog, with actor set $A = \{\text{system developer}, \text{scrum master}\}$ and a corresponding mandate set $M = \{(\text{system developer}, (d, 2)), (\text{scrum master}, (d, 1))\}$, where $d$ is a decision to prioritizing the backlog. In this $\mathsf{DCB}(C)$ the scrum master has less mandate in decisions regarding the backlog. An extension to this context would be to add tuple $(\text{product owner}, (d, 3))$ indicating that in this particular context the product owner has the right to final decisions of the backlog over the rest of the team. The final $\mathsf{DCB}(C)$ would then be: $\mathsf{DCB}(C) = \{d, \perp\}$ over the sets $A$ and $M$.

We note that for an organization $\mathsf{ORG}$, we can describe the complete decision landscape by:

$$\mathsf{DCB}_{\mathsf{ORG}} = \bigcup_{i=1}^{n} \mathsf{DCB}(C_i) \tag{2}$$

where $\mathsf{DCB}_{\text{org}}$ represents the set of all valid decisions across all contexts $\{C_1, C_2, \ldots, C_n\}$ in the organization.

Note that mandate sets are not exclusive, i.e. for an actor $a_k \in A_k$ that belongs to $C_k$ may have decision authority for a decision in multiple contexts. For example, given a decision $d \in \mathsf{DCB}(C_i)$ but also $d \in \mathsf{DCB}(C_k)$ the actor $a_k$ may have decision mandates $m_i$ in $C_i$ and $m_k$ in $C_k$, and $m_i \neq m_k$. This captures a managerial role that can have limited decision power in one context for $d$ but full mandate in another context.

## 4.2 Decision Conflicts

In the case where different actors have the same mandate level $m_i$, say $(a_1,(d,m_i)) \in M_k$ and $(a_2,(d,m_i)) \in M_k$ there might be possible conflicts in the decision-making process. For a subset $C_k = (A_k, M_k)$ to a context $C$, we define the *decision conflict function* $f : C_k \times D \to D$ that either render a decision $d \in D$ or the no decision $\perp \in D$. This function represents the process where a decision is made (thus to be executed in a context), including mandate conflicts.

**Definition 6** (Decision Conflict Function)**.** *Let $C_k = (A_k, M_k)$ where $A_k \subseteq A$ and $M_k \subseteq M$ for a certain context $C = (A, M)$. Let $f : C_k \times D \to D$ be function defined as:*

$$f(C_k,d) = \begin{cases} d_\delta & \text{if } \exists a_i, a_j \in A_k \text{ s.t } (a_i,(d,m_i)) \in M_k, \\ & (a_j,(d,m_j)) \in M_k \text{ and } m_i = m_j \\ d & \text{if } \exists a_k \in A_k \text{ s.t } (a_k,(d,m_k)) \in M_k \\ & \text{and } m_k > m_i \text{ for all } i \neq k \end{cases} \tag{3}$$

*where $d_\delta$ is a particular decision $d \in D$ with probability $\delta$ and has the possibility to yield $\perp$.*

Note that the decision mandate hierarchy holds trivially when no conflicts arise. The probabilistic outcome $d_\delta$ represents that *if* there is a conflict, it can be resolved in multiple ways. One such example is when an actor outside the current $\mathsf{DCB}(C)$ may interfere and make a decision that is not normally within the context decision space (e.g., upper management). In these cases, the likelihood of $d_\delta$ resolving to $\perp$ or a specific decision, depends on the used conflict resolution strategy or external organizational policies. Hence, it is up to the analyzing organization to define the probability distribution for $\delta$ associated to each possible $d$. We illustrate this function as follows: in ORG it is concluded that for a certain context $C_k = (A_k, M_k)$ there is a risk of decision conflict for decision $d$. Given the current policies and empirical knowledge, it is estimated that $f(C_k,d) = d'$ is given by $\Pr[d = \perp] = 0.5$ and $\Pr[d' = d] = 0.5$, i.e. the decision resolves to either no decision or the desired decision $d$ with a uniform distribution. Therefore $\delta = 0.5$ when resolving $d$ into $d'$, and similarly $\delta = 0.5$ when resolving $d$ into no decision.

To summarize, to model a particular decision outcome, we use the decision conflict function $f(C_k,d) = d'$ such that $d' = d$, $d' = d_\delta$ for some $d_\delta \in D$ with probability $\delta$, or $d' = \perp$.

## 4.3 Relation Definitions

We already defined the $\rightleftharpoons$ and $\rightharpoonup$ relations, but we also need several more to fully describe decision flows. In this section we defined four additional relations in the model, derived from the key insights.

### 4.3.1 Enforced Decisions

It might happen that a decision $d$ for which it holds that $d \rightleftharpoons C_1$ in $\mathsf{DCB}(C_1)$ where $C_1 = (A_1, M_1)$, but is enforced and executed by $a_k$ that belongs to $A_2$ in $C_2 = (A_2, M_2)$ with corresponding $\mathsf{DCB}(C_2)$. Regardless if $d \rightharpoonup C_2$ or not due to violations to the constraints in Def. 4 it might be the case the decision is executed anyway. This could happen if the decision-maker has high mandates in many other decisions (but not necessarily in $d$) or enforce the decision by also violating organizational constraints. As an example, a very high influencing employee could manipulate or make decisions without anyone noticing. In order to capture a violating decision $d$ into another context $C$ where $d$ does not belong, we denote that by the $\rightharpoondown$ relation:

**Definition 7.** *Let $d$ be a decision such that $d \notin \mathsf{DCB}(C)$ due to violation of any of the constraints in Def. 4. Then if $d$ is enforced by actor $a_k$ in $\mathsf{DCB}(C)$, we denote that relation as $d \rightharpoondown (C, a_k)$.*

### 4.3.2 Decision Dependencies

Decision dependencies represent situations where a decision $d$ in one context $C_1$ is dependent upon another decision $d'$ in context $C_2$ ($C_1$ can be the same as $C_2$ as well). These dependencies often arise in software engineering organizations when interconnected teams or units must align their decisions to achieve shared objectives, such as a development team depending on an architectural decision to proceed. Moreover, from key insight $\mathsf{KI}_5$ we conclude the need for such relation in the model. Capturing these relationships with the relation $d \leftarrow d'$ provides a way to model and analyze the interplay between decisions within and across contexts.

**Definition 8.** *Let $d \in C_1, d' \in C_2$ be two decisions in any two contexts $C_1$ and $C_2$. If there exists a dependency between $d$ and $d'$, i.e. $d$ cannot be made before*

*$d'$ is decided, we denote that relation as $d \leftarrow d'$ where the left-hand side is the dependent decision.*

### 4.3.3 Feedback-Driven Decisions

From the key insights we identified an reoccurring theme of decisions that required several feedback loops before execution ($\mathsf{KI}_3$). Especially when the information and/or knowledge of the decision-maker was lacking, such feedback loop is needed. Therefore, due to its common nature, we introduce the relation of a *decision requiring feedback*, denoted $d \circlearrowright a_k$ where $a_k$ is the actor that is requested for feedback. If a decision $d$ cannot be decided upon due to lack of information, it corresponds to this relation.

**Definition 9.** *Let $d$ be a decision for which $d \rightharpoondown a_k$ due to the knowledge constraint, we denote that relation as $d \circlearrowright C$ if there is a request of retrieving more information for decision $d$ from actor $a_k$.*

### 4.3.4 Decision for Escalation

A typical relation identified in the study were escalated decisions ($\mathsf{KI}_4$), namely that an actor could not make a decision within its own context, hence escalated the decision to someone with a higher mandate level (often a manager or architect). In a sense, this relation occurs trivially in any organization with authority levels, and aligns with Hackman's theory. This relation is naturally cross-context and we define it as follows:

**Definition 10.** *Let $d$ be decision for which $d \rightharpoondown C$. If the decision is escalated to another actor $a_k$ we denote that relation as $d \uparrow a_k$.*

## 4.4 Summary of Relations

A relation in the model translates to a state, namely a state within a context where a particular outcome is about to be reached. For example, $d \rightleftharpoons C$ indicates that a decision $d$ will be successful if executed within $C$ (not that $d$ was successfully executed). Similarly, $d \rightharpoondown C$ indicates that $d$ will not be able to be successfully executed in $C$, and $d \uparrow a_k$ indicates that $d$ is going to be escalated to $a_k$ as a next step. All defined relations are summarized in Tab. 3.

## 4.5 Model Syntax

To describe decision-making processes using the proposed model, we introduce a syntax that captures the flow of decisions within organizational contexts. This syntax includes both operators and symbols to represent sequential, conditional, and iterative flows, as well as termination states. The syntax is pseudo code inspired, thus universally understandable. A relation or a combination of relations and/or symbols and/or operators is called an *expression* and is denoted $\langle exp \rangle$. The following operators and symbols are defined for constructing decision flow expressions:

Table 3: Summary of decision relations in the proposed model.

| Notation | Relation/Construct |
|---|---|
| $d \rightleftharpoons C$ | Valid decision in context $C$ |
| $d \rightharpoondown C$ | Invalid decision in context $C$ |
| $d \neg (C, a_k)$ | Enforced decision into context $C$ by actor $a_k$ where $d \notin \mathsf{DCB}(C)$ |
| $d \circlearrowright a_k$ | Decision requires feedback from actor $a_k$ |
| $d \uparrow a_k$ | Decision escalated to actor $a_k$ |
| $d \leftarrow d'$ | Dependency for $d \in C$ to $d' \in C'$ |
| $\perp$ | No decision |

**START:** Represents the initial condition or state of the decision-making process, which sets a starting point of the flow and provides the starting context.

Example: START ($d \rightharpoondown C_1$) THEN $d \circlearrowright a_k$ UNTIL $d \rightleftharpoons C_1$.

**THEN:** Represents a sequential flow, where one relation follows another.

Example: $d \rightharpoondown C_1$ THEN $d \circlearrowright a_k$.

**IF:** Represents a conditional branch based on constraints or conditions.

Example: IF $d \leftarrow d'$ THEN $d' \rightleftharpoons C_2$.

**OTHERWISE:** Represents an alternative branch when the condition in IF is not satisfied.

Example: IF $d \leftarrow d'$ THEN $d' \rightleftharpoons C_2$ OTHERWISE $d' \rightharpoondown C_2$.

**UNTIL:** Represents an iterative flow that continues until a specific condition is met.

Example: $d \circlearrowright a_k$ UNTIL $d \rightleftharpoons C$.

**AND:** Represents parallel dependencies, where multiple relations must hold simultaneously.

Example: $d_1 \rightleftharpoons C_1$ AND $d_2 \circlearrowright a_k$.

**OR:** Represents alternative paths where at least one relation must hold.

Example: $d_1 \circlearrowright a_k$ OR $d_1 \uparrow a_m$.

**NOT:** Represents the negation of a relation or condition. We use the equivalence relation $\equiv$ to explicitly show a negated relation.

Example: NOT $d \rightleftharpoons C$ indicates that $d$ cannot be valid made in context $C$. Can also be expressed as NOT $d \rightleftharpoons C \equiv d \rightharpoondown C$.

**CONFLICT:** Represents a case of a decision conflict that occurs. This implicitly indicates the need for computing the decision conflict function.

Example: $d$ CONFLICT $(A, M)$ indicates that $d$ has a conflicting situation given actors in $A$ with mandates specified in $M$.

**RESOLVE:** Represents when the decision conflict function $f$ is used for yielding a decision outcome given a conflict.

Example: $d$ RESOLVE $(A, M) \rightarrow d_\delta$ indicates that $d$ was resolved into $d_\delta$, where $d_\delta$ is determined by a given probabilistic function specific to the organization.

**END:** The end of a decision flow is indicated using END, together with either $d \rightarrow \square$ (successful execution) or $d \leftarrow \perp$ (no decision).

Example: $d \rightleftharpoons C$ THEN $d \rightarrow \square$ END.

Example: $d \rightharpoonup C$ THEN $d \rightarrow \perp$ END.

**Parentheses:** Used to group sub-expressions and define precedence in decision flows. Follows the conventional rules for parentheses for logic expressions.

Example: $d \rightleftharpoons C_1$ THEN (IF $d \leftarrow d'$ THEN $d' \rightleftharpoons C_2$ OTHERWISE $d' \rightharpoonup C_2$).

To illustrate the usage of the syntax, consider a scenario where a decision $d_A$ is needed in context $C_A$ but is invalid due to information constraints, requires feedback from actor $a_k$, escalates to actor $a_m$ if unresolved, and concludes with either successful execution or no decision:

```
START ($d_A \rightharpoonup C_A$)
        THEN ($d_A \circlearrowright a_k$ UNTIL $d_A \rightleftharpoons C_A$)
        OTHERWISE ($d_A \uparrow a_m$)
        THEN IF $d_A \rightleftharpoons C_A$ THEN $d_A \rightarrow \square$
        OTHERWISE $d_A \rightarrow \perp$
        END.
```

Another example illustrates the process where a decision $d_A$ (e.g. architectural decision in a team) is not valid due to a dependency to $d_B$ (e.g. architect team approval):

```
START ($d_A \rightharpoonup C_A$)
        AND ($d_A \leftarrow d_B$)
        THEN ($d_B \uparrow a_B$ UNTIL $d_B \rightleftharpoons C_B$)
        THEN $d_B \rightarrow \square$
        THEN $d_A \rightleftharpoons C_A$
        THEN $d_A \rightarrow \square$ END.
```

These examples show the compactness and build up to sub expressions within a complete expression for an analyzed process. As seen in both examples, the THEN operator works as an implication and represents a logical next step in the process. This combined with a VSM, where details on why certain expressions are reached, would thus give a compact yet comprehensive mapping of a decision flow.

## 4.6 Axiomatic Rules

To ensure logical consistency in the model, we also introduce a set of fundamental rules. These are considered valid in the model given the caveat that they also assume a simplified worldview, e.g. strict sequential steps in a process and no detailed consideration to temporal aspects of a process. We use the standard mathematical notation of $\implies$ to show that a certain expression $\langle \exp \rangle_1$ logically leads to another expression $\langle \exp \rangle_2$, namely: $\langle \exp \rangle_1 \implies \langle \exp \rangle_2$.

**Definition 11** (Dependency Resolution Rule)**.** *A dependent decision cannot proceed unless its prerequisite decision is completed.*

$$d \leftarrow d' \implies (d \rightharpoonup C \text{ } OR \text{ } d' \rightharpoonup C')$$

*This ensures that decision d cannot be valid in context C unless the dependent decision $d'$ is valid in context $C'$.*

**Definition 12** (Feedback Resolution Rule)**.** *A decision requiring feedback must resolve its feedback loop before proceeding.*

$$(d \circlearrowright a_k \text{ } UNTIL \text{ } d \rightleftharpoons C) \implies d \rightharpoonup C \text{ } OR \text{ } d \circlearrowright a_m$$

*where $a_m$ is another feedback actor. This ensures that a decision in a feedback loop cannot proceed to validity until the necessary feedback is addressed.*

**Definition 13** (Escalation Termination Rule)**.** *An escalated decision must either reach a valid state or terminate with $\perp$.*

$$d \uparrow a_k \implies (d \rightleftharpoons C' \text{ } OR \text{ } \perp)$$

*This ensures that escalation leads to a resolution, either by validating the decision in a higher context $C'$ or terminating without resolution.*

**Definition 14** (Mutual Exclusion Rule)**.** *A decision cannot be escalated and resolved within the same context at the same time (simplifying temporal aspects), thus we have the implication that.*

$$d \rightleftharpoons C \implies NOT \text{ } d \uparrow a_k$$

*for any actor $a_k \in A$ where $C = (A, M)$.*

Table 4: Summary of the decision flow syntax, where ⟨exp⟩ is any valid expression constructed from the model.

| Syntax | Description |
|---|---|
| START ⟨exp⟩ | Specifies the initial condition of the decision process. |
| ⟨exp⟩ THEN ⟨exp⟩ | Sequential flow, where one expression follows another. |
| IF ⟨condition⟩, THEN ⟨exp⟩ | Conditional branch based on a specified condition. |
| OTHERWISE ⟨exp⟩ | Specifies the alternative path if the condition in IF is not met. |
| ⟨exp⟩ UNTIL ⟨condition⟩ | Iterative flow that continues until a specific condition is satisfied. |
| ⟨exp⟩ AND ⟨exp⟩ | Parallel dependencies, where both expressions must hold. |
| ⟨exp⟩ OR ⟨exp⟩ | Alternative paths, where at least one expression must hold. |
| NOT ⟨exp⟩ | Negation of an expression or condition. |
| ⟨exp⟩ THEN $\square$ END | Indicates the decision process ends with successful execution. |
| ⟨exp⟩ THEN $\perp$ END | Indicates the decision process ends with no decision. |
| Parentheses () | Groups sub-expressions and defines precedence. |

**Definition 15** (Loop Consistency Rule). *A decision $d$ that depend on decision $d'$ cannot at the same time have a dependency relation for $d'$; either $d$ is dependent on $d'$ or vice versa, otherwise there will be an infinite loop of dependency.*

$$d \leftarrow d' \implies NOT\ d' \leftarrow d$$

We acknowledge that these axiomatic rules are not complete nor exclusive; in this preliminary work we provide a first attempt to construct the model which needs to be further tested and evaluated.

## 4.7 Propositions and Explanations

In this section we summarize the developed model's propositions and associated explanations, based on the nomenclature and theory presentation structure in (Sjøberg et al., 2008). These are summarized in Tab. 5 and Tab. 6. Explanations are associated to the propositions of the theory, i.e. the model. The rationale is to have a sufficiently good motivation to the formulated proposition.

## 4.8 Scope

The scope of the model is that it is designed to capture and analyze decision-making processes within software development organizations, using a compact and efficient syntax, as a supportive tool in VSM activities. The model includes the capture of hierarchical, cross-context, and dependency-related challenges. While the model is grounded in the software engineering domain, its principles could possibly be generalized to other knowledge-intensive fields with similar organizational dynamics. The model's primary scope is thus to complement VSM activities and thereby provide insights into decision-making constraints, conflicts, and dependencies, enabling organizations to identify and address inefficiencies in their decision landscapes.

Table 5: Formulated propositions for the developed model.

| ID | Proposition |
|---|---|
| $P_1$ | Explicitly modeling decision dependencies and constraints enhances the the structural and relational understanding of decision-making processes. |
| $P_2$ | Using pseudo-code syntax enables systematic mapping and analysis of decision flows across multiple organizational contexts, supporting consistent representation of diverse decision processes. |
| $P_3$ | Using actor sets, mandate sets, and context boundaries with pseudo-code syntax allows for a compact and precise representation of decision processes, reducing ambiguity in process analysis. |
| $P_4$ | Using authority, knowledge, and resource constraints, along with mandate sets, enables the model to identify factors impacting decision validity. |
| $P_5$ | Integrating the model with VSM approaches supports the identification of process waste in decision-making workflows. |
| $P_6$ | Using explicitly defined relations such as dependencies, feedback loops, escalations, enforced decisions, systematically captures both linear and non-linear decision-making flows. |

## 5 DISCUSSION

In our model, Hackman's framework is used as basis for the design of the mandate set $M$ by highlighting the hierarchical relationships and distribution of decision-making power within and between contexts. For example, contexts operating under shared control often experience decision conflicts that require resolution mechanisms, which our decision conflict function $f$ addresses. In our model we can provide a more

Table 6: Explanations for the propositions of the developed model.

| ID | Explanation |
|---|---|
| $\mathsf{E}_1$ | Explicitly modeling decision dependencies and constraints provides an overview of th whole process. This is particularly important for addressing scenarios like $\mathsf{KI}_5$, where decisions are delayed due to unresolved dependencies, however, all insights $\mathsf{KI}_1$-$\mathsf{KI}_5$ indicate the need for a structured way to model a decision path. |
| $\mathsf{E}_2$ | Using pseudo-code syntax introduces a standardized representation of decision flows, making it easier to analyze, replicate, and communicate decision-making processes across different contexts. Such syntax is natural in software engineering, and allows for potential usage in other formal methods or tools for automatic syntax validation. |
| $\mathsf{E}_3$ | By defining actors and mandates as sets, with context boundaries, provide a structured (mathematical) way to model decision landscapes by capturing roles, responsibilities, and constraints in decision-making, since these are fundamental concepts in organizational structures. |
| $\mathsf{E}_4$ | Authority, knowledge, and resource constraints are key factors that influence decision validity, as motivated by $\mathsf{KI}_1$-$\mathsf{KI}_3$. By integrating these constraints, the model thus enables the identification of barriers in the decision-making process. |
| $\mathsf{E}_5$ | Integrating the model with VSM supports the identification of process waste, such as delays, hinders and unnecessary steps, by using the compact notation of the model with the practical application of VSM approaches. This explanation draws futher on lean management principles that emphasize waste reduction through process analysis. |
| $\mathsf{E}_6$ | Explicitly defined relations, such as dependencies, feedback loops, escalations, and enforced decisions, allow for the systematic representation of both linear and non-linear decision flows, capturing the complexity of real-world processes. This explanation is justified by $\mathsf{KI}_1$-$\mathsf{KI}_6$. |

granular view on the decision mandates compared to Hackman's framework, since we define sets of decisions with corresponding pre-specified mandate levels $m_i$. This aligns well with proposition $\mathsf{P}_4$ due to its simple nomenclature.

The foundation of bounded rationality directly connects to the proposed relations in the model, such as feedback loops and escalation, since it highlights how decision-makers operate with the different constraints. For example, this is captured in $\mathsf{P}_4$, showing the need to identify factors impacting decision validity when actors cannot exhaustively evaluate all options. Similarly, Hackman's theory has indeed shaped the model's constructs, such as decision context boundaries, which relates directly to the hierarchical decision-making processes described in $\mathsf{P}_3$ and $\mathsf{P}_6$. The escalation relation explicitly reflects Hackman's classifications of authority by demonstrating how decisions transition from self-control to managerial control when lower-level actors lack the authority (or has other constraints) to proceed. By integrating these theories, the model thus provides a structured way to formalize the interplay between decisions and decision-makers in a software engineering organization. As shown in the developed constructs and relations, it indeed provides a compact language to describe the decision flows (illustrated with the examples in Sec. 4.5), hence it would be supportive in VSM activities. Especially to describe the decision flows for both comparison and formal analysis within the same organization. The model is yet to be evaluated in single, but preferably multi-case studies, to show how well the support of the model gives a decision landscape VSM. This is our working hypothesis for future work, and concludes proposition $\mathsf{P}_5$.

The proposed model could have potential for automation in decision-support systems, supporting even more efficient VSM approaches. From this formalizing of decision-making processes with clearly defined constructs, rules, and relations, the model could potentially be implemented algorithmically to analyze workflows, identify bottlenecks, and suggest optimizations in real-time. Additionally, its pseudo-code-inspired syntax and logical structure would make it suitable for integration with tools for formal reasoning, enabling automated validation of decision flows, conflict detection, and different type of complex patterns. Moreover, considering different type of use cases where the model could be useful, the model syntax and rules most likely allow for both simple and more complex ones such as cross-organizational (thus cross-contextual) decision flows between different actors.

## 5.1 Example Scenario

We illustrate the syntax with another example. A project manager ($a_p$) needs to allocate cloud resources for a new microservice ($d_X$), but this requires prior technical approval for security compliance from the infrastructure team ($a_i$). The decision-making process is captured in steps: firstly, $a_p$ attempts to make a decision to allocate resources without approval ($d_X \rightharpoonup C_X$) and is identified to depend on a compliance decision ($d_X \leftarrow d_Y$). The decision is escalated to the infrastructure team ($d_Y \uparrow a_i$) which makes the decision ($d_Y \rightleftharpoons C_Y$). If the infrastructure team provides approval, the decision is successfully made ($d_Y \rightarrow \Box$) and the project manager is now able to proceed ($d_X \rightleftharpoons C_X$). The final decision is successfully made: $d_X \rightarrow \Box$. However, if the infrastructure team identifies security concerns, a feedback loop is initiated where the project manager must address the issues before resubmitting the request. If the concerns remain unresolved, the decision process may result in a failure ($d_X \rightarrow \perp$). The full description in the model is:

```
START ($d_X \rightharpoonup C_X$)
        AND ($d_X \leftarrow d_Y$)
        THEN ($d_Y \uparrow a_i$ UNTIL $d_Y \rightleftharpoons C_Y$)
        THEN ($d_Y \rightarrow \Box$)
        THEN $d_X \rightleftharpoons C_X$
        THEN $d_X \rightarrow \Box$ END.
            OR ($d_Y \rightarrow \perp$) IF unresolved.
```

From this analysis, it is possible to complement a VSM in resource allocation decisions to better understand the overall workflow with dependencies and parties.

## 5.2 Threats to Validity

This model is subject to certain limitations that may affect both its validity and applicability. The formalization make several assumptions that may not fully capture real-world scenarios, e.g. static actor- and mandate sets that does not fully reflect the dynamic nature of roles and responsibilities which frequently change. Moreover, while the model captures key decision relations such as escalation and feedback, it simplifies complex interactions like resource- and knowledge dependencies, which may require additional constructs. Also, the model builds on logical rules which assumes deterministic behavior, thus the current model does not include human factors, such as biases or conflicts that influence decision-making. However, this would complicated the model significantly, and could instead be captured in the VSM method. Finally, the sampling for the empirical data collection is limited due to its small size and respondents volume, however, it was collected from three different companies including both private and public sector.

## 5.3 Future Work

We are currently collaborating with one Swedish company to evaluate the model, aiming for a case study to further validate and refine the model. For future work we suggest a multi-case study with several different companies, thus potentially different decision processes, to map using VSM and the model. For more efficient analysis we also suggest that research into automation of VSM analysis based on our model should be done, including development of tools and/or complementary support systems. Most importantly, the model itself should be further analyzed and developed as in evaluating if additional rules or syntax is needed. Moreover, extensions to our work could be utility theoretic approaches where decision outcome and prediction could be additional parameters to consider.

## 6 CONCLUSIONS

We have provided a mathematical framework, derived from empirical data on real-world decision-making processes in software engineering organizations, to be used for better analysis and optimization of organizational decision-making. The model can express decision-making processes with a pseudo code syntax and expressions based on relations between decisions, actors and contexts. The model allows for identification of mandate hierarchies, decision flows and decision boundaries.

The proposed model is presented with the description of constructs, propositions with explanations, and scope, all according to theory presentation by (Sjøberg et al., 2008). Additionally, we also defined a model syntax and an axiomatic rule set within the model. From the propositions $P_1$-$P_6$, we conclude that an answer to RQ1 is provided, since the model shows that it can indeed be used for describing decision flows in accordance to VSM (or any other value flow technique for that matter). Although not extensively tested in practice, the theoretical framework is defined. The compact notation is illustrated with examples. We have contributed with a theoretical model for describing and analyzing decision-making flows in software engineering organizations, which can support managers with organizational work-flow or decision-making optimizations.

## REFERENCES

Botero, D., Monsalve, E. S., and Pardo, C. (2024). Practices for conducting value stream traceability in devops: A systematic literature mapping. *Periodicals of Engineering and Natural Sciences (PEN)*, 12(3):541–564.

Burrows, M., Abadi, M., and Needham, R. (1990). A logic of authentication. *ACM Transactions on Computer Systems (TOCS)*, 8(1):18–36.

Eleftherios Andreadis, J. A. G.-R. and Kumar, V. (2017). Towards a conceptual framework for value stream mapping (vsm) implementation: an investigation of managerial factors. *International Journal of Production Research*, 55(23):7073–7095.

Erbas, C. and Erbas, B. C. (2009). Software development under bounded rationality and opportunism. In *2009 ICSE Workshop on Software Development Governance*, pages 15–20.

Hackman, J. (1986). The psychology of self-management in organizations. *Psychology and Work: Productivity Change and Employment/American Psychological Association*.

Helleno, A., Pimentel, C., Ferro, R., Santos, P., Oliveira, M., and Simon, A. (2015). Integrating value stream mapping and discrete events simulation as decision making tools in operation management. *The International Journal of Advanced Manufacturing Technology*, 80:1059–1066.

Huang, Z., Kim, J., Sadri, A., Dowey, S., and Dargusch, M. S. (2019). Industry 4.0: Development of a multi-agent system for dynamic value stream mapping in smes. *Journal of Manufacturing Systems*, 52:1–12.

Kumar, R. (2025). A comprehensive review of mcdm methods, applications, and emerging trends. *Decision Making Advances*, 3(1):185–199.

Kunigami, M., Kikuchi, T., and Terano, T. (2019). *A Formal Model of Managerial Decision Making for Business Case Description*, pages 21–26. Springer Singapore, Singapore.

Liu, Q. and Yang, H. (2020). An improved value stream mapping to prioritize lean optimization scenarios using simulation and multiple-attribute decision-making method. *IEEE Access*, 8:204914–204930.

Mäkinen, S. (2024). Enhancing software development processes through value stream mapping–a case study. Master's thesis.

Marković, u., Vandevelde, S., Vanbesien, L., Vennekens, J., and Denecker, M. (2024). An epistemic logic for modeling decisions in the context of incomplete knowledge. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, SAC '24, page 789–793, New York, NY, USA. Association for Computing Machinery.

Mendes, E., Rodriguez, P., Freitas, V., Baker, S., and Atoui, M. A. (2018). Towards improving decision making and estimating the value of decisions in value-based software engineering: the value framework. *Software Quality Journal*, 26:607–656.

Meudt, T., Metternich, J., and Abele, E. (2017). Value stream mapping 4.0: Holistic examination of value stream and information logistics in production. *CIRP Annals*, 66(1):413–416.

Moe, N. B., Aurum, A., and Dybå, T. (2012). Challenges of shared decision-making: A multiple case study of agile software development. *Information and Software Technology*, 54(8):853–865.

Moe, N. B., Šmite, D., Paasivaara, M., and Lassenius, C. (2021). Finding the sweet spot for organizational control and team autonomy in large-scale agile software development. *Empirical Software Engineering*, 26(5):101.

Murphy, G. C. and Kersten, M. (2020). Towards bridging the value gap in devops. In Bruel, J.-M., Mazzara, M., and Meyer, B., editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 181–190, Cham. Springer International Publishing.

Pretorius, C., Razavian, M., Eling, K., and Langerak, F. (2024). When rationality meets intuition: A research agenda for software design decision-making. *Journal of Software: Evolution and Process*, page e2664.

Qin, Y. and Liu, H. (2022). Application of value stream mapping in e-commerce: a case study on an amazon retailer. *Sustainability*, 14(2):713.

Ralph, P. (2018). The two paradigms of software development research. *Science of Computer Programming*, 156:68–89.

Razavian, M., Paech, B., and Tang, A. (2019). Empirical research for software architecture decision making: An analysis. *Journal of Systems and Software*, 149:360–381.

Salin, H., Rybarczyk, Y., Han, M., and Nyberg, R. G. (2022). Quality metrics for software development management and decision making: An analysis of attitudes and decisions. In *International Conference on Product-Focused Software Process Improvement*, pages 525–530. Springer.

Simon, H. A. (1955). A behavioral model of rational choice. *The quarterly journal of economics*, pages 99–118.

Sjøberg, D. I., Dybå, T., Anda, B. C., and Hannay, J. E. (2008). Building theories in software engineering. *Guide to advanced empirical software engineering*, pages 312–336.

Tankhiwale, S. and Saraf, S. (2020). Value stream mapping (vsm) led approach for waste and time to market reduction in software product development process. *Telecom Business Review*, 13(1):27.

Wallsten, T. S. (2024). *Cognitive processes in choice and decision behavior*. Taylor & Francis.

Wang, Y., Liu, D., and Ruhe, G. (2004). Formal description of the cognitive process of decision making. In *Proceedings of the Third IEEE International Conference on Cognitive Informatics, 2004.*, pages 124–130. IEEE.

Yang, S., Nachum, O., Du, Y., Wei, J., Abbeel, P., and Schurmans, D. (2023). Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*.