Towards a Conceptual Model on Causal Loop Diagram

Júlia de Souza Borges[1], **Thiago Felippe Neitzke Lahass**[1], **Amanda Brito Apolinário**[1], **Monalessa P. Barcellos**[1]

[1] Ontology & Conceptual Modeling Research Group (NEMO)
Computer Science Department, Federal University of Espírito Santo
Av. Fernando Ferrari s/n, Campus de Goiabeiras – 29.060-900 – Vitória – ES – Brazil

{julia.s.borges, thiago.lahass, amanda.apolinario}@edu.ufes.br,

monalessa@inf.ufes.br

Abstract. Context: *Information systems are increasingly complex, shaped by interdependent technical, organizational, and human factors. Understanding this environment is crucial for developing and using information systems effectively. Systems Thinking (ST) offers tools that help in this matter. In ST, a system comprises elements and interconnections coherently organized to produce characteristic behaviors. ST tools aid in modeling complex systems and improving the understanding of their dynamics. Causal Loop Diagrams (CLDs), one of the most used ST modeling tools, represent cause-and-effect relationships and feedback structures, supporting the identification of problem causes, behavioral patterns, and enabling informed decision-making.* ***Problem:*** *Despite its potential, the lack of knowledge and supporting tools has limited CLD adoption.* ***Solution:*** *To structure knowledge about CLD, we developed a conceptual model representing its main elements, relationships, and constraints.* ***IS Theory:*** *This work relates to the organizational knowledge creation theory. A conceptual model provides an abstract representation of a domain, serving as a theoretical foundation that captures essential entities, their properties, and relationships. It enables knowledge formalization and supports communication among stakeholders.* ***Method:*** *To evaluate the model clarity and correctness, it was assessed by domain experts. To demonstrate its applicability, it was used to develop a supporting tool for CLD creation and interpretation.* ***Results:*** *Findings indicate that the conceptual model adequately captures essential concepts for understanding CLDs and provides structured knowledge useful for building them. The model can also serve as a reference model to aid tool development and has potential applications in semantic annotation and interoperability.* ***Contributions and Impact to IS area:*** *For researchers, this work advances knowledge about CLD, providing a formal representation of its conceptualization. For practitioners, it offers a model that aids in CLD understanding and communication, and in developing computational solutions. It also offers a tool to support CLD creation and interpretation, fostering its use in practice.*

1. Introduction

Information systems can be understood as organized arrangements of people, processes, and technologies designed to collect, process, store, and disseminate data, transforming it into information that supports individual, organizational, and social activities

[Checkland 1988]. Their hybrid nature involves both technical (e.g., hardware, software, and infrastructure) and social and behavioral subsystems, marked by dynamic and often ill-structured interactions [Antonelli et al. 2013]. This complexity implies that information systems cannot be analyzed solely from a reductionist or purely technical perspective, since their effectiveness is directly related to the human, organizational, and cultural dimensions in which they are embedded. Therefore, it becomes essential to adopt a broader and integrated perspective capable of understanding the interdependence between technical and social elements.

Systems Thinking (ST) offers a conceptual lens for addressing these challenges, as it emphasizes the identification of feedback structures, causal relationships, and recurring behavioral patterns over time [Meadows 2008]. In contrast to reductionist approaches in information system development, which treat organizational, technical, and human factors in isolation, ST highlights the interdependency among those fragments. This systemic view helps uncover the underlying structures that generate persistent problems and reveals leverage points for more effective interventions. By shifting the focus from individual events to the systemic drivers of organizational behavior, ST enables a deeper understanding of complexity and long-term dynamics [Fatema and Sakib 2017].

Within the ST perspective, an information system is viewed as a *system*, composed of elements (e.g., teams, artifacts, policies, technologies) and interconnections (e.g., the relationships among the development team, the software artifacts it produces, and the organizational policies influencing their production) that are coherently organized into a structure producing characteristic behaviors. These behaviors can often be interpreted as the system's function or purpose. For instance, an information system used to support decision-making in alignment with organizational objectives illustrates this systemic integration [Meadows 2008, Sterman 2010]. By addressing the interconnections among the several elements that comprise a system, ST enables a holistic view and provides a lens to understand complex interactions and anticipate the long-term consequences of decisions across technical and organizational dimensions.

The application of ST is particularly relevant in the information system context. ST provides modeling tools, such as Causal Loop Diagram (CLD), that support the visualization of cause-and-effect relationships between organizational processes, user interactions, and technological components, helping researchers and practitioners to capture systemic factors that influence quality, productivity, and alignment, and to explore how local decisions propagate into broader organizational consequences.

CLD is one of the ST tools organizations have used. A literature mapping study [Borges et al. 2024] revealed that it has been the predominant ST modeling tool in software development. However, although ST modeling tools have great potential to offer a comprehensive view of complex systems, facilitate the analysis of systemic interactions, and identify leverage points, their practical use remains limited. This underuse is partly due to the specialized knowledge required, which often involves a steep learning curve. The lack of knowledge is aggravated by the absence of a shared conceptual foundation and standardized terminology for CLD, leading to inconsistent interpretations and limiting their broader adoption. Furthermore, the lack of supporting tools devoted to CLD creation and interpretation represents a barrier that makes the practice more difficult [Borges et al. 2024]. A conceptual model would help address this gap by providing structured knowledge and a consolidated conceptualization to support understanding and communication about CLD. In our investigation, we did not find any study presenting such a model.

Conceptual models play a central role in scientific and engineering disciplines as they provide a structured and abstract representation of a domain. They serve as a theoretical foundation that captures essential entities, their properties, and relationships, enabling the formalization of knowledge and supporting communication among stakeholders. Conceptual modeling extends beyond diagrammatic notations, aiming to establish ontological commitments that reduce ambiguity and ensure semantic precision. Such models not only support understanding but also create a foundation for building computational tools, validating reasoning, and guiding systematic practices. In this sense, conceptual models act as bridges between informal mental models and formal system specifications, ensuring that domain knowledge is both human-comprehensible and machine-interpretable [Mylopoulos et al. 2025].

Given the discussion above, we developed a conceptual model representing the CLD core conceptualization. The *Conceptual Model on Causal Loop Diagram* helps human understanding and communication on the subject, reducing ambiguity and semantic conflicts. Moreover, the model can be used in model-driven approaches, aid in the development of computational tools and diagram data annotation.

To assess the model clarity and correctness, it was evaluated by two domain experts. The results indicate that the model is correct, clear, and covers the essential elements to understand and build CLDs. Additionally, to demonstrate the model's applicability, it was used in a case study to develop a tool that supports CLD creation and interpretation [Apolinário et al. 2025]. The tool allows creating CLDs, automatically detects feedback loops and archetypes, and prevents the user from making modeling mistakes. The conceptual model was used to develop the tool's structural model, and the constraints formalized in the conceptual model were used to implement the rules to ensure CLD modeling consistency and correctness.

This work contributes by offering a conceptual model that represents the CLD conceptualization, providing a foundation for communication and shared understanding, and defining a structural basis to guide the development of tools and practical applications involving CLD. The conceptual model allows researchers and practitioners to explore the use of CLD to model complex information systems, understand their behaviors, and make well-informed decisions.

From an Information Systems perspective, this contribution is grounded on the role of conceptual modeling as a fundamental mechanism for understanding and analyzing complex socio-technical systems, which involve the interaction of organizational processes, human actors, and technological components [Boscarioli et al. 2017]. In this context, modeling approaches are essential to support communication, reduce ambiguity, and enable systematic reasoning in the design and analysis of information systems.

This paper is organized as follows: Section 2 provides the background for the paper; Section 3 illustrates an application scenario that is used latter to exemplify concepts; Section 4 presents the *Conceptual Model on Causal Loop Diagram*; Section 5 regards the model evaluation by domain experts; Section 6 regards its use to develop a tool; Section

7 discusses related work; and Section 8 presents our final considerations.

2. Background

Systems Thinking can be understood as an integrated set of analytical skills that enable individuals to identify and comprehend systems, anticipate their behaviors, and design interventions to achieve desired outcomes. Rather than functioning as isolated techniques, these skills operate synergistically, reinforcing one another and forming a coherent approach to reasoning about complexity [Arnold and Wade 2015].

ST provides a lens to understand complexity by focusing on how elements interrelate within a system rather than analyzing them in isolation. A system is not merely a collection of parts, but a set of interconnected elements that, together with their environment, form a unified whole oriented toward a purpose [Drack and Apfalter 2007, Meadows 2008]. From this perspective, system behaviors and outcomes emerge from the dynamics of the whole, making it insufficient to analyze parts separately [Checkland 1999, Kim 2018].

Originating from Systems Theory, ST has been applied in diverse domains such as biology, management, and engineering, providing conceptual foundations and a variety of modeling tools [Sterman 2010, Sterman 1994]. In the context of information systems development, ST has been used mainly to understand organizational and project dynamics, supporting actions for process and product improvement, particularly in Requirements Engineering and Project Management [Borges et al. 2024].

ST has a set of tools that provide complementary perspectives. For example, Stock & Flow Diagrams emphasize resource accumulation, Influence Diagrams highlight decision consequences, and CLDs uncover the causal relationships and interaction patterns that shape complex systems [Burge 2015, Sterman 2010].

Among these, CLD has emerged as one of the most prominent [Borges et al. 2024], as it captures cause-and-effect relationships and feedback structures essential to the inherently complex interdependence of software projects, shaped by interrelated technical, organizational, and human factors [Meadows 2008]. At the same time, CLD provides a lightweight and intuitive entry point for ST, bridging technical and managerial perspectives, and making explicit how requirements, quality, cost, and time constraints interact and evolve over time [Ciancarini. et al. 2023, Huang and Fang 2022, Senapathi and Drury-Grogan 2021].

A CLD depicts the dynamics of a system by identifying its boundaries, relevant variables, and the causal relationships among them. These causal links can be positive or negative. A positive causal relationship indicates that two variables change in the same direction (e.g., an increase in design flaws raises the number of defects), while a negative causal relationship indicates that the variables change in opposite directions (e.g., increasing testing effectiveness reduces defects) [Meadows 2008]. By representing these causalities explicitly, CLDs allow analysts to visualize how local actions may ripple through the system and generate broader impacts.

Central to CLD are the Feedback Loops, which capture how changes propagate within the system. There are two types. Balancing loops counteract variation, acting as stabilizing mechanisms that promote equilibrium and resistance to change (e.g., defect reduction practices that keep defect rates under control). Reinforcing loops, in turn, amplify change in one direction, creating dynamics of escalation or decline (e.g., schedule pressure leading to rushed work, which increases defects and further amplifies pressure). These loops are responsible for much of the non-linear behavior observed in complex systems [Sterman 2010].

A key advantage of CLDs is the possibility of identifying Archetypes, which are recurring system structures associated with typical behavioral [Meadows 2008]. For instance, Shifting the Burden is an archetype that emerges when short-term symptomatic fixes delay fundamental solutions, and Fixes that Fail manifests through interventions that generate unintended long-term consequences [Kim and Anderson 1998]. The ability to recognize these archetypes is particularly valuable in information systems, where complex socio-technical systems are the norm. Understanding these recurring patterns supports better decision-making, promotes proactive identification of leverage points, and enables the design of sustainable interventions. This systemic perspective is essential for managing project dynamics, improving organizational processes, and addressing root causes rather than symptoms [Clancy 2018].

CLDs have been applied to different phenomena. For example, they have been used to analyze requirement changes and rework in development processes [Andersson et al. 2002], to evaluate agile and lean practices [Ching and Mutuc 2018], to investigate factors influencing team productivity [Fatema and Sakib 2017], and to support Kanban adoption [Senapathi and Drury-Grogan 2021].

3. Illustrative Application Scenario

Before presenting the *Conceptual Model on Causal Loop Diagram*, we describe a scenario that illustrates its concepts. Our goal is to provide the reader with a real-world scenario in which CDLs were applied and use this scenario later as an example to instantiate the model's concepts. The scenario is detailed in [dos Santos Júnior et al. 2022]. In that study, CLDs were applied to support the implementation of agile practices in a software organization. The Brazilian organization, referred to here as Organization A, collaborates with a European partner, Organization B, to deliver software products to clients abroad. Organization B elicits requirements from clients, and Organization A is in charge of developing the corresponding software. As a consequence of the increasing number of projects and team members, added to the lack of flexible processes, some problems emerged, such as projects being late and over budget, an increase in software defects, overloading of the teams due to rework on software artifacts, and communication issues among client, Organization A, and Organization B.

Aiming to minimize these problems, Organization A decided to implement agile practices. A Systems Thinking-based approach was employed to identify the practices suitable for the organization and the leverage points that the practices should focus on. The first steps were to understand the organization and build a systemic view. For that, the researchers used CLDs. Figure 1 illustrates a fragment of a CLD developed in the study [dos Santos Júnior et al. 2022]. The elements in blue in the figure form a modeling pattern that reveals the presence of the archetype Shifting the Burden. Next, we briefly describe the organizational scenario represented in the CLD.

As previously said, Organization B is responsible for eliciting requirements from the client, specifying and sending them to Organization A to develop the software. The development teams of Organization A often misunderstand requirements that describe the software, component, or functionality to be developed, since Organization B produces *Requirements poorly specified*, neither adopting a proper technique nor following a pattern to describe them. *Misunderstood requirements* contribute to increasing the number of *Defects in Software Artifacts*, since design, code, and test are produced based on the requirements informed by Organization B. *Defects in Software Artifacts* make Organization A mobilize (and often overload) the development team to fix defects by performing *New urgent development activities*, which decrease the number of *Defects in Software Artifacts*. These urgent activities are performed as fast as possible, aiming not to delay other activities. Thus, they do not properly follow the best practices in software quality. Moreover, they contribute to increasing the project cost and time (*Late and over-budget project*). *Defects in Software Artifacts* increase the need for *Use of Software Quality Techniques* that, when used, lead to fewer *Defects in Software Artifacts* and also delay their occurrence over time.

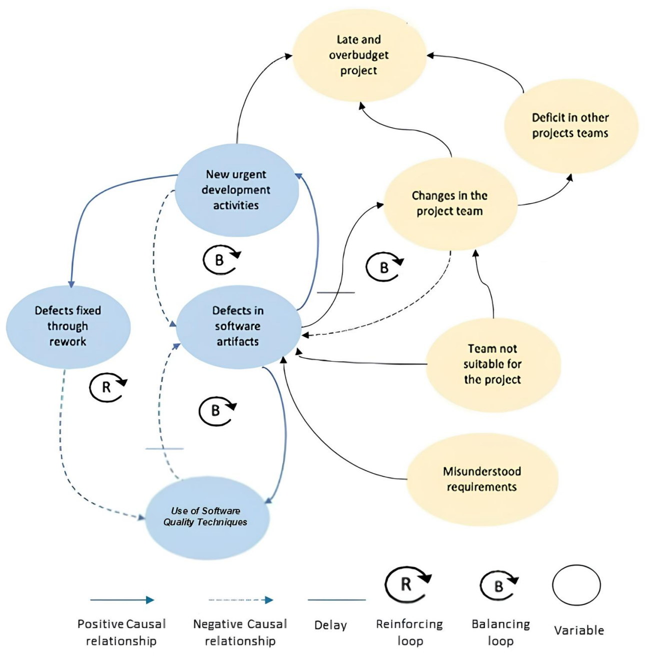

Late and overbudget project
Deficit in other projects teams
New urgent development activities
Changes in the project team
B
B
Defects fixed through rework
Defects in software artifacts
Team not suitable for the project
R
B
Misunderstood requirements
Use of Software Quality Techniques
Positive Causal relationship
Negative Causal relationship
Delay
R
Reinforcing loop
B
Balancing loop
Variable

Figure 1. Fragment of CDL (adapted from [dos Santos Júnior et al. 2022])

The elements in blue in the Figure 1 highlight a modeling pattern that reveals the presence of the archetype Shifting the Burden, including three balancing feedback loops and one reinforcing feedback loop.

4. A Conceptual Model on Causal Loop Diagram

To represent the conceptual model, we used a UML class diagram in which the concepts corresponding to entities are illustrated as classes, and their properties are specified as attributes. The conceptual model was developed to represent the concepts necessary to build a CLD and also identify the modeling patterns that arise from the CLD and reveal specific behaviors (i.e., archetypes). Figure 2 presents the conceptual model. Next, we describe it. In the text, **bold** is used for concepts, <u>underline</u> for attributes and *italics* for instances.

Additionally, the model description includes axioms to address constraints not explicitly captured in the diagram and represent modeling patterns that identify archetypes.

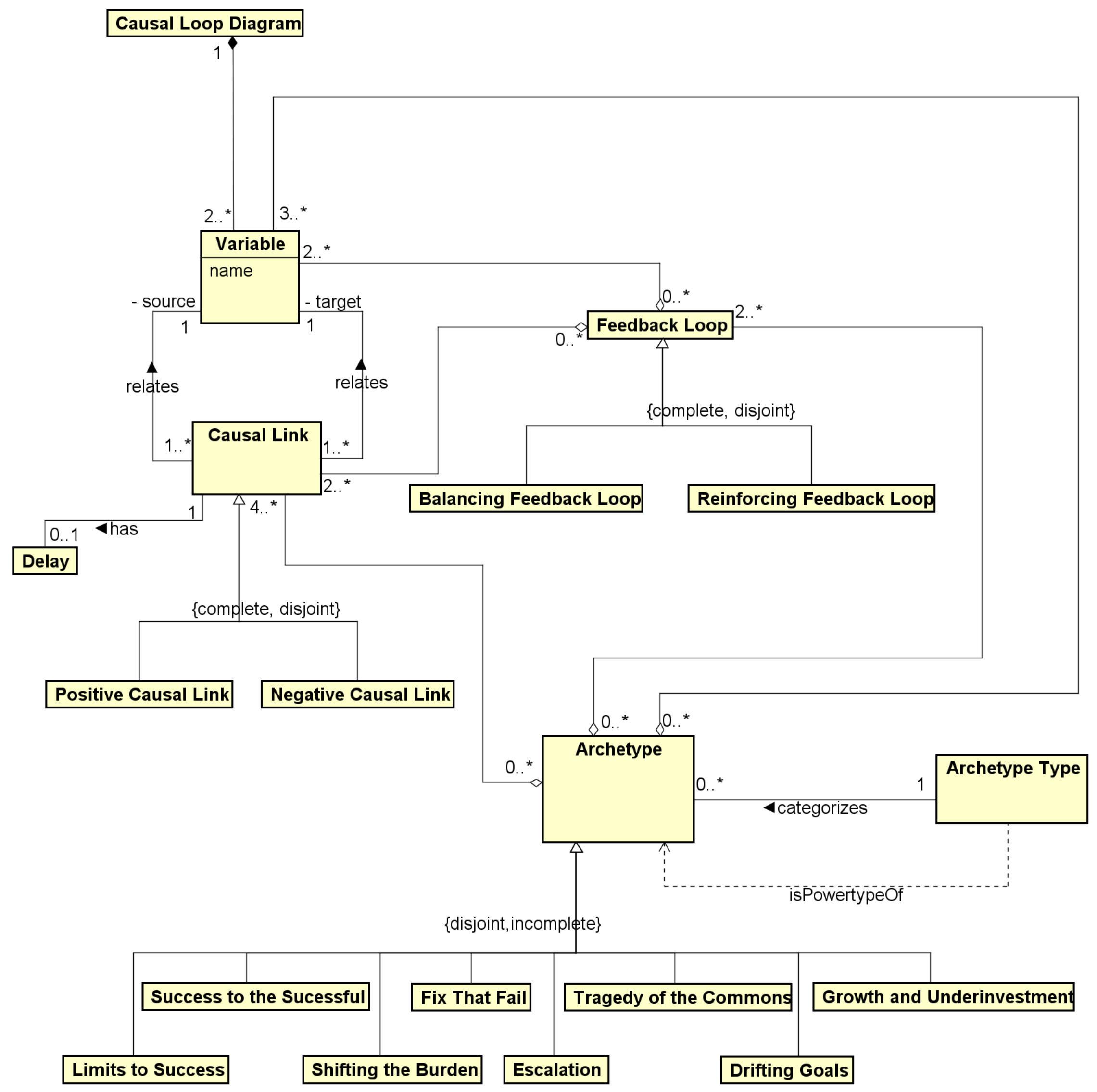


Figure 2. Causal Loop Diagram Conceptual Model

The **Causal Loop Diagram** concept serves as a container for all the elements related to a CLD. A **Causal Loop Diagram** is composed of **Variables** connected to each other by arrows (relationships) that indicate **Causal Links** between the related **Variables**. **Variables** represent factors that change over time and influence the system's behavior [Sterman 2010]. Each **Variable** has an attribute called name that identifies the **Variable** as unique. This uniqueness is essential to avoid ambiguity in causal relationships and ensure model consistency. In the example presented in Section 3, variables are represented in ellipses. For example, *Use of software quality techniques* and *Defects in software artifacts* are **Variables**, each with a distinct name to ensure uniqueness.

In a CLD, each **Variable** is connected to another through a **Causal Link** that defines the relationship between the two **Variables** [Sterman 2010]: the **source** and the

target. In a **CLD**, a **Variable** cannot be related to itself. Moreover, the **Causal Link** that connects two **Variables** that are part of a **CLD** is also part of that **CLD**. These constraints are represented in First Logic Order, respectively, in the axioms A1 and A2.

A 1. *∀ v1, v2 Variable, c, CausalLink (relates (c, v1) ∧ source (v1) ∧ relates (c, v2) ∧ source (v2) → (v1 ≠ v2))*

A 2. *∀ cld CausalLoopDiagram, v1, v2 Variable, c, CausalLink (relates (c, v1) ∧ (relates (c, v2) ∧ partOf (v1, cld) ∧ partOf (v2, cld)) → partOf (c, cld)*

A **Causal Link** can be a **Positive Causal Link** or a **Negative Causal Link**. A **Positive Causal Link** represents a direct relationship in which an increase (or decrease) in the source **Variable** leads to an increase (or decrease) in the target **Variable**, moving them in the same direction [Sterman 2010]. In the example, the **Variable** *Defects in software artifacts* (source) is linked to *Use of software quality techniques* (target) by a **Positive Causal Link** (solid arrow connecting these variables), meaning that an increase/decrease in *Defects in software artifacts* increases/decreases the need for *Use of software quality techniques*. A **Negative Causal Link**, in turn, represents an inverse relationship where an increase (or decrease) in the source **Variable** leads to a decrease (or increase) in the target **Variable**, moving them in opposite directions [Sterman 2010]. For instance, the **Variable** *Use of software quality techniques* (source) is linked to *Defects in software artifacts* (target) by a **Negative Causal Link** (dotted arrow connecting these variables), which means that an increase/decrease in the use of *Use of software quality techniques* leads to a decrease/increase in *Defects in software artifacts*.

In a **CLD**, if a **Positive Causal Link** relates the **Variables** *v1* (source) and *v2* (target), then there is no **Negative Causal Link** in that **CDL** relating *v1* (source) and *v2* (target). Similarly, if a **Negative Causal Link** relates the **Variables** *v3* (source) and *v4* (target), then there is no **Positive Causal Link** in the **CLD** relating *v3* (source) and *v4* (target). That is, in a **CLD**, two **Variables** cannot be connected by different types of **Causal Links** in the same direction. These constraints are presented in the axioms A3 and A4 below.

A 3. *∀ v1, v2 Variable, c1, c2: PositiveCausalLink (relates (c1, v1) ∧ source(v1) ∧ relates (c1, v2) ∧ target(v2)) → ¬∃ c2: NegativeCausalLink (relates (c2, v1) ∧ source(v1) ∧ relates (c2, v2) ∧ target(v2))*

A 4. *∀ v1, v2 Variable, c1, c2: NegativeCausalLink(relates (c1, v1) ∧ source(v1) ∧ relates(c1, v2) ∧ target(v2)) → ¬∃ c2: PositiveCausalLink (relates (c2, v1) ∧ source(v1) ∧ relates(c2, v2) ∧ source(v2))*

A **Causal Link** can have a **Delay**, which indicates that the effects of changes on the source **Variable** take time to affect the target **Variable** [Meadows 2008] and this can influence the system's behavior. For example, consider the **Negative Causal Link** between the **Variables** *Use of software quality techniques* (source) and *Defects in software* *artifacts* (target). An increase in the *Use of software quality techniques* will only lead to a reduction in defects after some time. Hence, there is a **Delay** (solid line crossing the dotted arrow between *Use of software quality techniques* and *Defects in software artifacts*) in this relationship.

Variables and **Causal Links** can form **Feedback Loops**. A **Feedback Loop** consists of at least two **Variables** interconnected by **Causal Links** in such a way that form a closed cycle that influences the system behavior over time. In the simplest case, a **Feedback Loop** occurs when a **Variable** *v1* (source) is related to the **Variable** *v2* (target) through a **Causal Link** *c1*, while *v2* (source) is related back to *v1* (target) through another **Causal Link** (*c2*). Together, *v1*, *v2*, *c1*, and *c2* form a closed structure of mutual influence, i.e., a **Feedback Loop**. For example, this is the case of the variables *Defects in software artifacts* and *Use of software quality techniques* and the two causal links that connect these variables in Figure 1. This is formally defined in First Logic Order in the axiom A5.

A 5. *∀ v1, v2: Variable, c1, c2: CausalLink (relates(c1, v1) ∧ source(v1) ∧ relates(c1, v2) ∧ target(v2)) ∧ (relates(c2, v2) ∧ source(v2) ∧ relates(c2, v1) ∧ target(v1)) → ∃ f: FeedbackLoop isPartOf(c1, f) ∧ isPartOf(v1, f) ∧ isPartOf(c2, f) ∧ isPartOf(v2, f)*

Although **Feedback Loops** are formally defined in their minimal structure in the axiom A5, consisting of two **Variables** connected by two **Causal Links** forming a closed cycle, this representation serves as a foundational case. **Feedback Loops** can involve more than two **Variables**, interconnected by a series of **Causal Links** that together form closed paths of influence. Depending on the configuration of the **Feedback Loop**, it can be classified as a **Balancing Feedback Loop** or a **Reinforcing Feedback Loop**.

A **Balancing Feedback Loop** aims to stabilize the system, regulating its behavior over time to reach a specific target value or remain within a predefined range. Thus, it acts against any change that attempts to push the system out of this equilibrium [Meadows 2008]. A **Balancing Feedback Loop** works to bring a system to a target state and keep it stable. It is characterized by an odd number of **Negative Causal Links**, which reverse the feedback's direction compared to the initial change [Kim 1993].

For example, in Figure 1, there is a **Balancing Feedback Loop** between *Defects in software artifacts* and *Use of software quality techniques*. When *Defects in software artifacts* (source) increase, the organization increases the *Use of software quality techniques* (target) to address the rising issues (there is a positive causal link between these variables). As the *Use of software quality techniques* (source) increases, it helps reduce *Defects in software artifacts* (target) (there is a negative causal link between these variables), gradually stabilizing the system by maintaining a balance between quality assurance efforts and defect prevention.

In a **Reinforcing Feedback Loop**, a change in the cause amplifies the effect, which then further reinforces the cause, creating a self-enhancing cycle. This type of loop can lead to exponential growth or rapid decline over time, as it continuously reinforces itself [Meadows 2008]. Structurally, a **Reinforcing Feedback Loop** is characterized by having an even number of **Negative Causal Links**, including the possibility of none. This configuration ensures that the overall polarity of the loop is reinforcing, allowing the initial change to propagate and escalate through the system without opposition [Sterman 2010].

In Figure 1 , there is a **Reinforcing Feedback Loop** involving the following **Variables**: *New urgent development activities*, *Defects in software artifacts*, *Use of software quality techniques*, and *Defects fixed through rework*. The increase of *Defects in software artifacts* leads to the increase of *New urgent development activities* to fix them, which contributes to increasing the number of *Defects fixed through rework* (there are positive Causal Links between these variables). As the number of *Defects fixed through rework* increases, the use of *Use of software quality techniques* decreases (because much effort is spent on rework and there is no time left for applying *Use of software quality techniques*). As a consequence of the reduction in the use of *Use of software quality techniques*, there is an increase in *Defects in software artifacts* (there are negative causal links between these variables). This loop reinforces a behavior that fixes defects through rework and hampers software quality.

In a **CLD**, if a set of **Variables** and their corresponding **Causal Links** form a **Reinforcing Feedback Loop**, then it is not possible for the same set of elements to simultaneously constitute a **Balancing Feedback Loop**. This mutual exclusivity arises because the polarity and dynamic behavior of each type of loop are fundamentally distinct. Therefore, if a **Balancing Feedback Loop** is identified using a given configuration of **Variables** and **Causal Links**, this configuration cannot also define a **Reinforcing Feedback Loop**. Axioms A6 and A7 illustrate these constraints for the simplest configuration of **Feedback Loops**, i.e., constituted by two **Variables** and two **Causal Links**.

A 6. ∀ *v1, v2: Variable, c1, c2: CausalLink, r: ReinforcingFeedbackLoop (relates(c1, v1) ∧ source(v1) ∧ relates(c1, v2) ∧ target(v2) ∧ relates(c2, v2) ∧ source(v2) ∧ relates(c2, v1) ∧ target(v1) ∧ isPartOf(v1, r) ∧ isPartOf(v2, r) ∧ isPartOf(c1, r) ∧ isPartOf(c2,r)) → ¬∃ b: BalancingFeedbackLoop (relates(c1, v1) ∧ source(v1) ∧ relates(c1, v2) ∧ target(v2) ∧ relates(c2, v2) ∧ source(v2) ∧ relates(c2, v1) ∧ target(v1) ∧ isPartOf(v1, b) ∧ isPartOf(v2, b) ∧ isPartOf(c1, b) ∧ isPartOf(c2,b))*

A 7. ∀ *v1, v2: Variable, c1, c2: CausalLink, b: BalancingFeedbackLoop (relates(c1, v1) ∧ source(v1) ∧ relates(c1, v2) ∧ target(v2) ∧ relates(c2, v2) ∧ source(v2) ∧ relates(c2, v1) ∧ target(v1) ∧ isPartOf(v1, b) ∧ isPartOf(v2, b) ∧ isPartOf(c1, b) ∧ isPartOf(c2,b)) → ¬∃ r: ReinforcingFeedbackLoop (relates(c1, v1) ∧ source(v1) ∧ relates(c1, v2) ∧ target(v2) ∧ relates(c2, v2) ∧ source(v2) ∧ relates(c2, v1) ∧ target(v1) ∧ isPartOf(v1, r) ∧ isPartOf(v2, r) ∧ isPartOf(c1, r) ∧ isPartOf(c2,r))*

An **Archetype** is a structure that represents a characteristic pattern of behavior. It is composed of a combination of **Variables** and **Causal Links** , constituting at least two **Feedback Loops** (thus, it has at least three **Variables** and four **Causal Links**). **Archetypes** help identify recurring dynamics and provide insights into managing or anticipating system behavior. Given that an **Archetype** is composed of **Feedback Loops**, which, in turn, are composed of **Variables** and **Causal Links**, and the part-of relation is transitive, the **Variables** and **Causal Links** that compose a **Feedback Loop** that is part of an **Archetype** are also part of that **Archetype**. Thus,

A 8. $\forall$ *v: Variable, f: FeedbackLoop, a: Archetype isPartOf(v, f)* $\land$ *isPartOf(f, a)* $\rightarrow$ *isPartOf(v, a)*

A 9. $\forall$ *f: FeedbackLoop, a: Archetype, c: CausalLink isPartOf(c, f)* $\land$ *isPartOf(f, a)* $\rightarrow$ *isPartOf(c, a)*

Archetypes are categorized in **Archetype Types** (e.g., **Limits to Success, Shifting the Burden, Fix That Fail, Success to the Successful, Escalation, Tragedy of the Commons, Drifting Goals, Growth and Underinvestment**). In Figure 2, **Archetype Type** is represented as a powertype of **Archetype**. Therefore, concepts specialized from **Archetype** represent instances of **Archetype Type** [Guizzardi et al. 2015]. Thus, **Limits to Success, Shifting the Burden, Fix That Fail, Success to the Successful, Escalation, Tragedy of the Commons, Drifting Goals** and **Growth and Underinvestment** are examples of **Archetype Types.** There are several **Archetype Types**. Each type of **Archetype** is characterized by a specific modeling pattern.

For example, **Shifting the Burden** is constituted of four **Variables**, two **Balancing Feedback Loops** (one of them with a **Delay**) and one **Reinforcing Feedback Loop**[1]. It represents a situation where a problem symptom prompts an intervention, and a quick, symptomatic solution is applied, temporarily easing the issue, and forming a **Balancing Feedback Loop**. However, this symptomatic solution diminishes the perceived need for a more fundamental, long-term intervention, leading to the creation of another **Balancing Feedback Loop** that further **delays** the resolution of the root cause. Over time, the symptomatic solution causes side effects that erode the capacity to apply the fundamental solution, creating a **Reinforcing Feedback Loop** where the effectiveness of the symptomatic solution decreases, and the ability to address the root cause declines [Kim and Anderson 1998]. This leads to a cycle where the problem recurs, and the fundamental solution becomes increasingly difficult to implement.

In Figure 1, the elements represented in blue reveal a **Shifting the Burden** archetype. In that context, recurring *Misunderstood requirements* lead to an increase in *Defects in software artifacts*. To deal with these defects, the organization often resorts to urgent corrective efforts by overloading the development team with *New urgent development activities*, which temporarily reduces the defects. However, these quick fixes typically bypass best practices in software quality, contributing to increased project costs and delays (*Late and over-budget projects*). In parallel, the presence of defects also drives the adoption of *Use of software quality techniques*, which, when applied, help reduce defect recurrence more sustainably. The Shifting the Burden archetype emerges as the organization repeatedly favors symptomatic solutions (urgent development) over fundamental ones (quality practices). This dynamic forms a reinforcing loop where temporary fixes mask the deeper issue, reducing the perceived need for structural improvements and ultimately perpetuating the cycle of rework and overload.

The modeling pattern that characterizes a **Shifting the Burden** archetype is

[1]Due to space limitations, in this paper, we do not detail the other types of archetypes.

captured by the axiom A10.

A 10. *∀ a: Archetype, d: Delay, b1, b2: BalancingFeedbackLoop, r: ReinforcingFeedbackLoop, v1, v2, v3, v4: Variables, c1, c2, c3, c4, c5, c6: CausalLink (relates(c1, v1) ∧ source(v1) ∧ relates(c1, v2) ∧ target(v2) ∧ relates(c2, v2) ∧ source(v2) ∧ relates(c2, v1) ∧ target(v1) ∧ isPartOf(v1, b1) ∧ isPartOf(v2, b1) ∧ isPartOf(c1, b1) ∧ isPartOf(c2, b1) ∧ relates(c3, v2) ∧ source(v2) ∧ relates(c3, v3) ∧ target(v3)) ∧ (relates(c4, v3) ∧ source(v3) ∧ relates(c4, v2) ∧ target(v2) ∧ has(c4, d) ∧ isPartOf(v2, b2) ∧ isPartOf(v3, b2) ∧ isPartOf(c3, b2) ∧ isPartOf(c4, b2) ∧ relates(c5, v1) ∧ source(v1) ∧ relates(c5, v4) ∧ target(v4) ∧ (relates(c6, v4) ∧ source(v4) ∧ relates(c6, v3) ∧ target(v3) ∧ (relates(c4, v3) ∧ source(v3) ∧ relates (c4, v2) ∧ target(v2) ∧ has(c4,d)) ∧ relates(c2, v2) ∧ source(v2) ∧ relates(c2, v1) ∧ target(v1) ∧ isPartOf(v1, r) ∧ isPartOf(v4, r) ∧ isPartOf(v3, r) ∧ isPartOf(v2, r) ∧ isPartOf(c4, r) ∧ isPartOf(c2, r) ∧ isPartOf(c5, r) ∧ isPartOf(c6, r) ∧ isPartOf(b1, a) ∧ isPartOf(b2, a) ∧ isPartOf(r, a)) → ShiftingTheBurden (a)*

5. Evaluating the Conceptual Model from the Domain Expert Perspective

To evaluate the proposed conceptual model, we conducted a study with domain experts to gather feedback regarding the model's clarity, correctness, and its potential to support the use of CLDs and the development of computational solutions.

Study Design

Aligned with the study's goal, we defined the following research questions: *Is the conceptual model correct and clear*? *Does the conceptual model have the potential to help the use of CLD and support the development of computational solutions*?

The **instruments** used in the study consisted of two main artifacts: (i) a document in PDF format containing the conceptual model specification, and (ii) a form created in Google Forms to collect the participants' perceptions. The conceptual model specification comprises the UML class diagram representing concepts and relationships, the model description (also including axioms), and a table summarizing the concepts' definition. The form is composed of a consent term for participation in the study and two sections of questions. The first section contains eight closed questions to characterize the participant. The second section is composed of eight questions related to the research questions. Seven are closed questions (adopting the Likert Scale), but the participant is asked to justify their answer. There is also an open question for the participant to provide general comments and suggestions.

The study's **participants** needed to be people with knowledge of and practical experience with Systems Thinking, particularly CLD, to evaluate the model under the domain expert perspective. Moreover, they needed to have knowledge of and practical experience with conceptual modeling to be able to evaluate the model's correctness.

The **procedure** followed in the study consisted of inviting people who fit the desired profile. Two people were invited and all accepted (hereinafter they are referred to as P1 and P2). The conceptual model specification and the evaluation form were sent to the participants. They were asked to read the specification and then fill in the evaluation form. To make it easier for the participants to provide detailed feedback, they were allowed to include additional comments directly in the conceptual model specification. The participants had two weeks to complete their assessment. Next, we summarize the main results. The study protocol, used instruments, and collected data are available in the study package at `https://doi.org/10.6084/m9.figshare.30131926`.

Study Results

Concerning the participants' profiles, both hold a Ph.D. in Computer Science and have worked as teachers and researchers at a federal institution of higher education for more than 10 years. In addition, both have worked on projects in partnership with the industry for more than 10 years. Both participants declared to have good knowledge and high practical experience with CLD (P1 has worked with this topic for 3 years and P2 for more than 5 years), and high knowledge and very high practical experience with conceptual modeling (both have worked with this topic for more than 5 years).

Regarding the *Conceptual Model on CLD* and the concepts definition, P1 and P2 agreed that "the conceptual model covers the concepts and relations necessary to represent CLDs". According to P2 "*The main concepts corresponding to the understanding and modeling of CLD, such as variable, link and loop, are covered in the conceptual model*". P1 and P2 also agreed (P2 strongly agreed) that "the conceptual model is correct (i.e., it represents concepts and relationships correctly)". P2 commented that "*the concepts and related descriptions presented correspond with the correct semantics of the constructs of CLD*". The participants also agreed (P2 strongly agreed) that "the conceptual model represents useful knowledge about CLD". P1 stated that "*the model is useful for those who are not yet familiar with the concepts of CLD*" and P2 said "*The conceptual model does help users better understand the CLDs, as it clarifies the meanings of the constructs, what a variable is, what a causal link is, etc.*". P2 highlighted that some constructs can be too abstract for the user and suggested "*the use of ontological distinctions*" to analyze them in depth. Both participants agree that "the definitions presented for the concepts are correct" and "the definitions presented for the concepts are clear". However, P1 pointed out that "*Variable was defined at a very high level*". P2 also mentioned that and reinforced that ontological distinctions could help make the definitions clearer (e.g., considering variables with different granularity).

When asked about the potential applications of the conceptual model, both participants agreed (P2 strongly agreed) that "the conceptual model can help understand CLDs". P2 stated that "*in addition to making the semantics clearer, it uses a notation familiar to software engineers (UML class diagram), making it even easier to understand*". P2 also strongly agreed that "*the conceptual model can help develop solutions (e.g., tools, data annotation, data integration)*" and said that "*the greater the understanding of the CLD constructs from the conceptual model, the better the use of this tool and the better the solutions given*". P1 was neutral about that and suggested that "*a guideline with examples and a step-by-step approach would be useful for applying the model*" to develop solutions.

Results Interpretation

The results suggest that the *Conceptual Model on CLD* effectively captures the essential concepts and relationships needed to represent CLDs. The participants judged it to be correct, accurately reflecting the relevant concepts and their interconnections. According to the participants, the definitions are clear and grounded in the literature, ensuring conceptual accuracy.

Both evaluators emphasized that the model supports the understanding of fundamental constructs, such as variables and causal links, reinforcing its value. Moreover, the adoption of UML class diagram notation was identified as a relevant design choice that connects abstract constructs to a notation already familiar to many system engineers.

The results also suggest that the model enhances both CLD comprehension and use: the clearer the understanding of CLD constructs, the more effectively system engineers can build CLDs and apply the conceptual model to develop computational solutions. These findings underscore the importance of combining theoretical rigor with practical applicability, suggesting that the model not only represents concepts accurately but can also provides value in real-world contexts.

Improvement opportunities were also identified. Both participants highlighted that the *Variable* concept seems to be very abstract. For example, it does no explicitly address aspects related to granularity to differentiate simple from complex variables. P2 suggested using ontologies to make these distinctions clearer. Although the proposed conceptual model properly addresses the necessary concepts and presents them correctly and clearly, the concepts could still be further analyzed. Having a conceptual model grounded on a foundational ontology could help classify the concepts in the ontology categories, deepen the understanding of the constructs and, consequently, of the created CLDs and the phenomenon they describe.

In summary, the model was considered correct, clear, and able to help the use of CLDs and support the development of computational solutions related to this subject. Moreover, there is still room to analyze the concepts in depth by grounding them in ontological foundations.

Study Limitations

As any study, this one has some limitations that must be considered together with the results. The main limitations are the small number of participants and the fact that they did not use the model in practice before assessing it.

Concerning the participants, although there are only two, they have a proper profile (academic background and practical experience) to ensure the credibility of the results. Both are researchers holding Ph.D. degrees in Computer Science and high experience in industry. Both have deep expertise in conceptual modeling and System Thinking, combining theoretical knowledge with practical experience. This combination of academic preparation and hands-on expertise enables the participants to understand the proposed conceptual model and be well-positioned to provide reliable feedback. It must be noted that such a profile is not very common.

The fact that the participants did not use the model before evaluating it must also be considered a relevant limitation. Some questions answered by the participants refer to the potential use of the model. Therefore, their answers indicate their perceptions about this potential and are not based on real applications of the model by the participants. Both are highly experienced with the use of conceptual models to build computational solutions (e.g., information systems, annotations, interoperability solutions). Therefore, we believe that even though they did not apply the model, the participants' profile allows them to provide valuable feedback. Even so, this limitation cannot be disregarded.

6. Applying the Conceptual Model to develop a CLD Supporting Tool

To complement the domain experts' evaluation and demonstrate the applicability of the *Conceptual Model on CLD*, it was used to develop CaLMo [Apolinário et al. 2025], a tool that supports the creation and interpretation of CLDs. The tool supports users to create CLDs, automatically identifies feedback loops and archetypes, and prevents the user from making modeling mistakes (e.g., contradictory polarities or circular references between variables). Additionally, the tool provides knowledge about CLD and guidance on how to create them.

The tool was developed by two software engineers (not the authors of the *Conceptual Model on CLD*), who used the *Conceptual Model on CLD* to create the tool's structural model (class diagram) by defining the entities (e.g., Variable, Feedback Loop), their properties, and their relationships. The axioms supported the implementation of domain-specific constraints to ensure consistency and correctness of the created CLDs. The use of the *Conceptual Model on CLD* provided domain knowledge that helped the developers understand the concepts involved in the addressed domain. Moreover, it accelerated the development process because the developers used the conceptual model as a basis instead of developing the tool's class diagram and domain-specific rules from scratch. Figure 3) presents the tool's structural model.

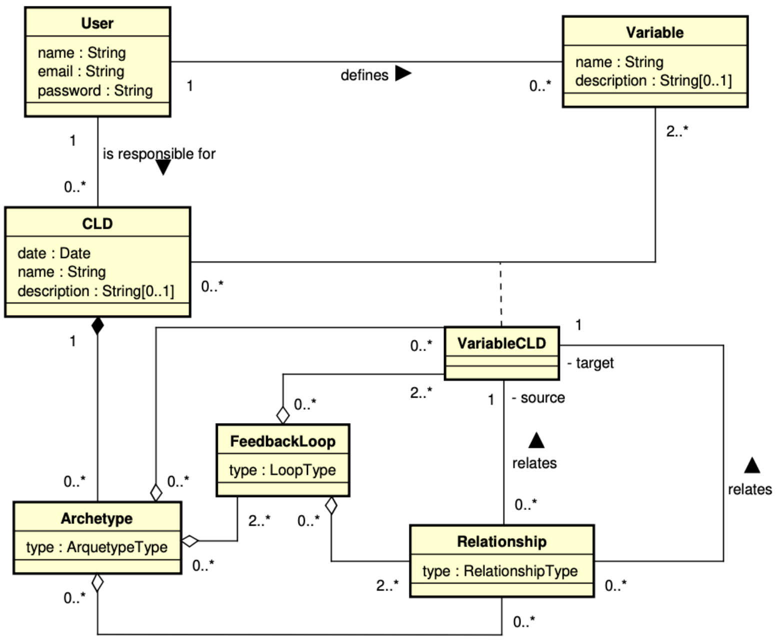


Figure 3. Class diagram of CaLMo [Apolinário et al. 2025]

The classes *CLD*, *Variable*, *Relationship*, *Feedback Loop*, and *Archetype* correspond respectively to the concepts **Causal Loop Diagram**, **Variable**, **Causal Link**, **Feedback Loop**, and **Archetype** from the *Conceptual Model on CLD*. The specializations of **Archetype**, **Causal Link**, and **Feedback Loop** were addressed in the class diagram as domain-specific enumerated types: *LoopType*, indicating whether a loop is Balancing or Reinforcing; *RelationshipType*, specifying whether a causal link is Positive or Negative; and *ArchetypeType*, which categorizes feedback structures into predefined types of archetypes such as Shifting the Burden and Fixes that Fail. A property *type* was added to the *Relationship*, *Feedback Loop*, and *Archetype* classes. These enumerations add semantic precision to the diagram, enabling the tool to apply consistency rules and support automated pattern recognition.

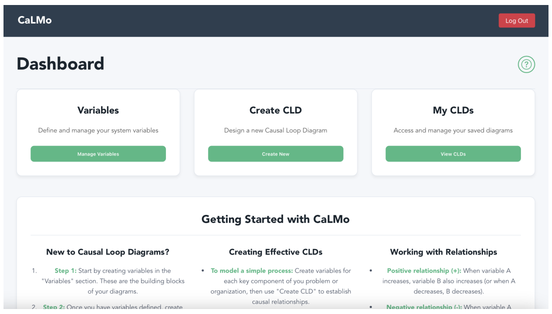
To enable the tool to control access to the created CLDs (a user can access only the CLDs created by them), the *User* class was added to the diagram.

The tool allows users to reuse a *Variable* in different *CLDs*. To manage the many-to-many relationship between a general *Variable* and a specific *CLD*, the association class *VariableCLD* was introduced. This class models the concrete instance of a variable within a particular CLD, serving as the actual node to which a *Relationship* is attached as its source or target.

Next, we briefly present the main features of CaLMo. A detailed description of the tool, including its architecture and functionalities, can be found in [Apolinário et al. 2025]. A short video demonstrating CaLMo is available at https://zenodo.org/records/15477387 and the source code repository can be found at https://zenodo.org/records/15476020. CaLMo is available at https://dev.nemo.inf.ufes.br/calmo/.

Figure 4 presents an overview of the CaLMo main page, which serves as the central navigation hub and provides direct access to the main features: *create variables*, *create CLDs*, and *visualize CLDs*. This page also offers access to tutorial resources that guide users through the tool's features and provide knowledge to help the user create and interpret CLDs.

To create a CLD, the user must create the variables and indicate the relationships between them (Figure 5). Thus, the tool automatically generates the CLD (Figure 6 shows the CLD produced for the illustrative scenario presented in Section 3). In the CLD, blue is used to highlight variables that form the Shifting the Burden archetype, which is automatically identified by the tool. The other variables appear in yellow. Positive causal links are represented in green, while the negative ones are in red. Users can customize the CLD by repositioning nodes to improve layout clarity.



Define and manage your system variables

Design a new Causal Loop Diagram

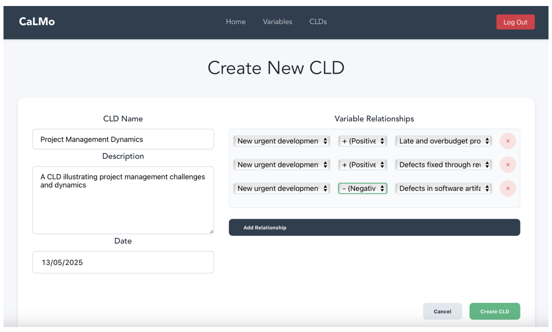
Access and manage your saved diagrams

1. **Step 1:** Start by creating variables in the "Variables" section. These are the building blocks of your diagrams.
2. **Step 2:** Once you have variables defined, create

- **To model a simple process:** Create variables for each key component of you problem or organization, then use "Create CLD" to establish causal relationships.

- **Positive relationship (+):** When variable A increases, variable B also increases (or when A decreases, B decreases).
- **Negative relationship (-):** When variable A

Figure 4. CaLMo main page



CLD Name: Project Management Dynamics

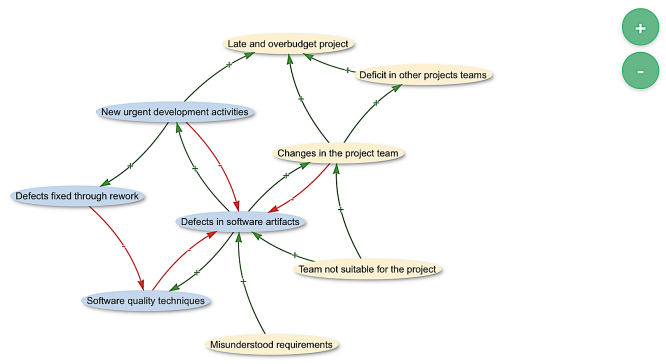
Description: A CLD illustrating project management challenges and dynamics

Date: 13/05/2025

New urgent developmen	+ (Positive	Late and overbudget pro
New urgent developmen	+ (Positive	Defects fixed through rev
New urgent developmen	- (Negativ	Defects in software artifa

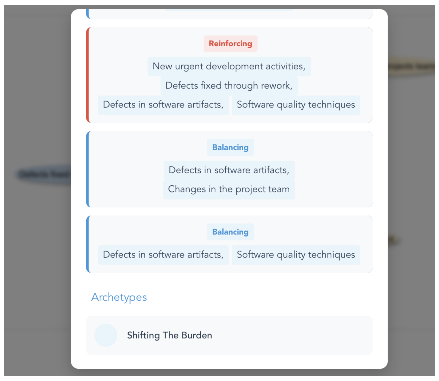
Figure 5. Establishing relationships between variables

CaLMo has analytical capabilities that extend beyond visualization. By clicking on variables, loops, or archetypes, the user accesses details about them (Figure 7), which can help analyze the CLD and understand the behaviors represented in it. For example, the user can examine a variable that has many relationships (such as "*Defects in software artifacts*" in the considered example).



Late and overbudget project
Deficit in other projects teams
New urgent development activities
Changes in the project team
Defects fixed through rework
Defects in software artifacts
Team not suitable for the project
Software quality techniques
Misunderstood requirements
+
-

Figure 6. Produced CLD



Reinforcing

New urgent development activities,
Defects fixed through rework,
Defects in software artifacts, Software quality techniques

Balancing

Defects in software artifacts,
Changes in the project team

Balancing

Defects in software artifacts, Software quality techniques

Archetypes

Shifting The Burden

Figure 7. Feedback loop and archetype details

7. Related Work

We did not find in the literature any work presenting a conceptual model on CLD. Hence, considering that our final purpose is to provide structured knowledge to support the use of CLD and aid in the development of computational solutions in this topic, in this section, we discuss some works that have proposed approaches for the creation, refinement, and formalization of CLD.

In [Binder et al. 2004], a structured process is proposed for transforming CLDs into system dynamics models, as a Stock and Flow Diagram. Their approach outlines how to identify key variables, establish causal links, and translate feedback loops into mathematical formulations that can be simulated. The study addresses CLDs for simulation purposes and leverages them as the initial stage of a rigorous and iterative method for building system dynamics models.

In [Dhirasasna and Sahin 2019], the authors present a multi-methodology approach for developing CLDs. The authors integrate quantitative techniques (e.g., stakeholder identification method, a systematic quantitative literature review, and MICMAC structural analysis) with traditional qualitative methods (e.g,. stakeholder interviews and engagement workshops). While maintaining the essence of the original CLD construction process, the proposed approach introduces new stages to define stakeholders, variables, their roles, and their interrelationships in a more quantitative and logically structured manner. This integration aims to reduce subjectivity, enhance the traceability of modeling decisions, and improve the overall robustness of the resulting CLDs.

Practical guidance for constructing CLDs has also been a topic of study. Works such as [Tip 2011], [Kim 1995], [Lawrence 2024], and [Haraldsson 2004], have offered guidelines to help create CLD (e.g., how to correctly indicate polarities and loop types).

While these contributions are relevant to support creating CLDs, none of them structure knowledge as a conceptual model. Thus, they do not establish a conceptualization able to support the understanding and communication about CLD. The proposed conceptual model increases the knowledge basis on CLD by clearly defining its main constructs and offering a reference model to build tools and other solutions. Developing such tools is important to promote the use of CLD. CLD users typically employ general-purpose diagramming or presentation tools (e.g., draw.io), which are not tailored to the particularities of CLD modeling. They lack mechanisms for dynamic identification of structures (e.g., feedback loops) and behavioral patterns (archetypes), polarity assignment, and do not provide means for verifying modeling rules or ensuring model consistency. This imposes additional effort to create CLDs and increases the risk of errors. In addition, most of these tools are limited to drawing the diagram and do not store the underlying data it represents. By providing a conceptual model that aid in understanding CLD constructs and developing computational solutions, this work helps bridge theoretical foundations, modeling practices, and practical applications.

8. Final Considerations

This work proposed a conceptual model on CLD. By formally defining fundamental constructs such as variables, causal links, feedback loops, and archetypes, the model clarifies the semantics of CLD and provides a structured foundation for its creation and analysis. Moreover, by systematizing knowledge about CLD, the conceptual model favors CLD adoption and contributes to a broader understanding of complex systems. Furthermore, it facilitates understanding among stakeholders and supports interoperability across different CLD modeling tools.

The evaluation by domain experts indicated that the model is correct and has the potential to support CLD understanding and practical applications. The use of the model to develop a supporting tool demonstrates the feasibility of translating the represented conceptualization into computational support. CaLMo, the developed tool, distinguishes itself from general diagramming tools – which have often been used to create CLDs – because it allows creating, storing, and managing CLDs, rather than just drawing graphical representations. Additionally, it automatically detects loops and archetypes and preserves semantic relationships for refinement.

Despite the promising results, this work has limitations, particularly regarding the limited number of experts involved in the evaluation and the use of the model to build a single tool. Future work includes new studies applying the proposed model for communication and understanding purposes, as well as to support the development of other computational solutions. Based on suggestions given by the domain experts who evaluated the model, we also intend to carry out an ontological analysis to understand CLD constructs in depth and refine abstract concepts such as *Variable*. Additionally, a framework for semantic interoperability among CLD diagramming tools can be structured based on the conceptual model. Finally, we have dedicated effort to produce guidelines to be associated with the conceptual model to offer a more comprehensive and practical guidance on CLD.

Acknowledgments

This research is supported by the Coordination for the Improvement of Higher Education Personnel - CAPES Brazil (Finance Code 001), the National Council for Scientific and Technological Development - CNPq (Process 304787/2025-6), and the Espírito Santo

Research and Innovation Support Foundation - FAPES (Processes 2023-5L1FC, 2022-NGKM5, 2021-GL60J, and T.O. 1022/2022). In this paper, Grammarly was used to correct writing errors and polish some parts of the text.

References

Andersson, C., Karlsson, L., Nedstam, J., Host, M., and Nilsson, B. I. (2002). Understanding software processes through system dynamics simulation: a case study. In *Proceedings Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pages 41–48. IEEE.

Antonelli, D., Chiabert, P., and Romagnoli, V. (2013). Information system and systems thinking: a compulsory marriage? *IFAC Proceedings Volumes*, 46(9):1780–1785. 7th IFAC Conference on Manufacturing Modelling, Management, and Control.

Apolinário, A., Lahass, T., Borges, J., Júnior, P. S., and Barcellos, M. (2025). Calmo: A tool to support the use of causal loop diagram in software engineering. In *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software*, pages 969–975, Porto Alegre, RS, Brasil. SBC.

Arnold, R. D. and Wade, J. P. (2015). A definition of systems thinking: A systems approach. *Procedia computer science*, 44:669–678.

Binder, T., Vox, A., Belyazid, S., Haraldsson, H., and Svensson, M. (2004). Developing system dynamics models from causal loop diagrams. In *Proceedings of the 22nd International Conference of the System Dynamic Society*, pages 1–21.

Borges, J., Lahass, T., Apolinário, A., Júnior, P. S., and Barcellos, M. (2024). Unveiling the landscape of system thinking modeling tools use in software engineering. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software*, pages 47–57, Porto Alegre, RS, Brasil. SBC.

Boscarioli, C., de Araujo, R. M., and Maciel, R. S. P. (2017). I grandsi-br grand research challenges in information systems in brazil 2016-2026.

Burge, S. (2015). The systems thinking tool box. *Burge, Hughes, Walsh*, pages 1–5.

Checkland, P. (1988). Information systems and systems thinking: Time to unite? *International Journal of Information Management*, 8(4):239–248.

Checkland, P. (1999). *Systems Thinking, Systems Practice*. John Wiley & Sons.

Ching, P. M. and Mutuc, J. E. (2018). Evaluating agile and lean software development methods from a system dynamics perspective. In *2018 IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pages 1–6. IEEE.

Ciancarini., P., Farina., M., Kruglov., A., Succi., G., and Thapaliya., A. (2023). A reflection on the use of systemic thinking in software development. In *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE*, pages 521–529. INSTICC, SciTePress.

Clancy, T. (2018). Systems thinking: Three system archetypes every manager should know. *IEEE Engineering Management Review*, 46(2):32–41.

Dhirasasna, N. and Sahin, O. (2019). A multi-methodology approach to creating a causal loop diagram. *Systems*, 7(3):42.

dos Santos Júnior, P. S., Barcellos, M. P., and Calhau, R. F. (2022). First step climbing the stairway to heaven model-results from a case study in industry. *Journal of Software Engineering Research and Development*, 10:5–1.

Drack, M. and Apfalter, W. (2007). Is paul a. weiss' and ludwig von bertalanffy's system thinking still valid today? *Systems Research and Behavioral Science: The Official Journal of the International Federation for Systems Research*, 24(5):537–546.

Fatema, I. and Sakib, K. (2017). Factors influencing productivity of agile software development teamwork: A qualitative system dynamics approach. In *24th Asia-Pacific Software Engineering Conference (APSEC)*, volume 2017-December, pages 737–742. IEEE.

Guizzardi, G., Almeida, J., Guarino, N., and Carvalho, V. (2015). Towards an Ontological Analysis of Powertypes. In *The Joint Ontology Workshops at the International Join Conference on Artificial Intelligence*.

Haraldsson, H. V. (2004). *Introduction to system thinking and causal loop diagrams*. Department of chemical engineering, Lund University Lund, Sweden.

Huang, T. and Fang, C.-C. (2022). Applying system dynamics approach for optimizing software release decisions. In *2022 2nd International Conference on Computation, Communication and Engineering (ICCCE)*, pages 54–57. IEEE.

Kim, D. (1993). System archetypes i: Diagnosing systemic issues and designing high-leverage interventions.

Kim, D. (2018). *Introduction to System Thinking*. Pegasus Communications.

Kim, D. H. (1995). *Systems thinking tools: a user's reference guide*. Toolbox reprint series. Pegasus Communications.

Kim, D. H. and Anderson, V. (1998). Systems archetype basics. *Waltham, Mass, Pegasus Communications Inc*.

Lawrence, M. (2024). Causal loop diagrams handbook. Technical Paper 2024-3, Cascade Institute.

Meadows, D. H. (2008). *Thinking in systems: A primer*. chelsea green publishing.

Mylopoulos, J., Guizzardi, G., and Guarino, N. (2025). Conceptual modeling: Foundations, a historical perspective, and a vision for the future. *Data & Knowledge Engineering*, page 102483.

Senapathi, M. and Drury-Grogan, M. L. (2021). Systems thinking approach to implementing kanban: A case study. *Journal of Software: Evolution and Process*, 33(4):e2322.

Sterman, J. (1994). Learning in and about complex systems. systems dynamics review, 10, 1994.

Sterman, J. (2010). *Business Dynamics: Systems Thinking And Modeling For The Complex World*. Tata McGraw Hill Education Private Limited.

Tip, T. (2011). Guidelines for drawing causal loop diagrams. *Systems Thinker*, 22(1):5–7.