CaLMo: A Tool to support the use of Causal Loop Diagram in Software Engineering

Amanda Brito Apolinário
Ontology & Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo
Vitoria, ES, Brazil
amanda.apolinario@edu.ufes.br

Thiago Felippe Neitzke Lahass
Ontology & Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo
Vitoria, ES, Brazil
thiago.lahass@edu.ufes.br

Júlia de Souza Borges
Ontology & Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo
Vitoria, ES, Brazil
julia.s.borges@edu.ufes.br

Paulo Sérgio dos Santos Júnior
LEDS Research, Department of Informatics, Federal Institute of Education, Science, and Technology of Espírito Santo
Serra, ES, Brazil
paulo.junior@ifes.edu.br

Monalessa P. Barcellos
Ontology & Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo
Vitoria, ES, Brazil
monalessa@inf.ufes.br

ABSTRACT

Software organizations involve several processes, people, practices, culture, and other factors that affect their behavior. Understanding the organizational environment is crucial for improving processes and products. System Thinking (ST) can help in this matter. It views an organization as a system composed of elements and interconnections coherently organized in a structure that produces a characteristic set of behaviors. ST offers several tools, among which the Causal Loop Diagram (CLD) stands out. It represents feedback-driven behaviors in complex systems and enables the visualization of organizational dynamics and the identification of systemic patterns. By understanding how the organization behaves, it is possible to identify problems and define actions aimed at improving products and processes. Currently, the creation of CLDs often relies on generic drawing/diagramming tools, which are prone to modeling errors and do not support the identification or analysis of behavior patterns. Additionally, most of these tools are limited to drawing the diagram and do not store the underlying represented data. To address this gap, this paper presents CaLMo, a tool designed to assist software engineers in modeling and analyzing CLDs to support Software Engineering tasks. CaLMo allows users to define variables, establish causal links between them, and detect feedback loops. Moreover, it automatically identifies archetypes that represent behavior patterns. We illustrate the use of the tool considering a case study in which CLDs were used to support the adoption of agile practices in an organization. A short video showing CaLMo is available at https://doi.org/10.5281/zenodo.15477387.

KEYWORDS

Causal Loop Diagram, System Thinking, Software Engineering, Modeling Tool

1 Introduction

Characteristics and demands of the modern and digital society have transformed the software development scenario and presented new challenges to software engineers, such as faster deliveries, frequent changes in requirements, lower tolerance to failures, and the need to adapt to contemporary business models [1]. Coping with these challenges is complex and involves various aspects, such as processes, people, tools, and culture. Hence, a broad and systemic view of the organizational environment is required. System Thinking (ST) is a suitable approach to provide such a view [2].

ST views an organization as a system[1], composed of interconnected elements (e.g., teams, artifacts, and practices) that work together to produce specific behaviors or outcomes [10, 12]. ST offers diagramming techniques and tools (e.g., Causal Loop Diagram and Stock-and-Flow Diagram) that help visualize, analyze, understand, and communicate system structures and behaviors, supporting the generation of insights and identification of effective interventions [6]. Here, we refer to these diagramming techniques and tools that support system modeling as *ST modeling tools*.

In a prior study [2], we conducted a systematic literature mapping to investigate the use of ST modeling tools in Software Engineering (SE). The results revealed that the *Causal Loop Diagram* (CLD) stands out as the most frequently used in SE, mainly due to its ability to illustrate cause-and-effect relationships and feedback loops. However, although ST modeling tools have shown promise in offering a holistic view of complex systems, supporting the understanding of systemic interactions, and identifying actions to mitigate problematic behaviors impacting SE outcomes, their use to address SE problems has been underexplored. One of the reasons is that using such tools often requires specific knowledge and may present a steep learning curve. Moreover, there is a lack of automated support to model and validate the diagrams [2].

In our investigation, we did not identify any tools specifically designed to support the modeling and analysis of CLDs. The analyzed studies typically employed general-purpose diagramming

[1]Thus, in this paper, when talking about systems in the ST context, we are referring to organizational environments (or extracts of them).

or presentation tools (e.g., Visual Paradigm[2] and draw.io[3]), 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. We also performed an informal search on the Internet, and found only one tool devoted to CLDs: LoopX [11] allows for the generation of CLDs from models created in XMILE format. The tool relies on pre-built XMILE models, does not detect archetypes, and targets simulation rather than CLD construction, failing to support CLD creation from scratch.

CLDs can be applied to support several tasks in SE. We highlight its application in the context of the Causal Analysis and Resolution (CAR) process area of the Capability Maturity Model Integration (CMMI), which aims at identifying the root causes of problems or defects and implementing effective corrective actions [7]. By explicitly representing causal relationships, CLDs can enhance this process, supporting diagnosis and continuous improvement. CLDs can also contribute to requirements engineering, enabling a holistic understanding of the software organization, the application domain, and how variables associated with them relate to each other.

Aiming to fill the gaps of existing tools and support the use of CLDs in SE, we have developed **CaLMo** (*Causal Loop diagram Modeler*). The tool allows users to define variables, establish causal links between them, and identify feedback loops. Additionally, it automatically identifies archetypes that represent behavior patterns.

This paper introduces CaLMo and is organized as follows: Section 2 provides a brief background; Section 3 presents an illustrative scenario to contextualize the use of the tool; Section 4 presents CaLMo architecture and conceptual model; Section 5 addresses CaLMo's features and presents some screenshots; Section 6 discusses the implications and contributions of the tool, and finally, Section 7 presents our final considerations.

2 Background

ST is based on System Theory. At its core, this theory emphasizes the need for a holistic understanding of systems, recognizing that system behavior cannot be fully explained by analyzing individual components in isolation [5]. Instead, systems must be understood as a set of interdependent elements whose interactions give rise to emergent, dynamic behaviors [10].

ST has been used in industry and academia for years [8, 10]. Its value lies in enabling the analysis of systemic structures and behavioral patterns that underlay persistent challenges. In Software Engineering, ST helps understand software organizations to identify actions aiming at process and product improvement. It has been used mainly in Requirements Engineering and Project Management contexts [2].

Among the ST modeling tools, the CLD has emerged as one of the most prominent [2]. A CLD consists of *variables* connected by *causal links*, which express the direction and type of influence between the variables. These links may be *positive*, denoting that changes in the source variable lead to changes in the same direction in the target, or *negative*, indicating an inverse relationship [8]. When connected in closed chains, these links form *feedback loops*, which are fundamental to understanding system behavior over time. There are two main types: *reinforcing feedback loops*, which amplify changes and may lead to exponential growth or decline, and *balancing feedback loops*, which counteract change and work toward system stability and equilibrium [10].

A significant advantage of using CLDs is their ability to uncover recurring behavioral structures known as *archetypes*, which are common system structures that produce characteristic patterns of behaviors [10]. They act as diagnostic templates that help detect systemic issues early and reason about their long-term consequences. Common examples include: *Shifting the Burden*, where quick fixes divert attention from fundamental solutions; *Fixes that Fail*, in which short-term interventions produce delayed negative side effects; and *Success to the Successful*, where unequal resource allocation reinforces disparities among system agents [9, 10].

The ability to recognize these archetypes is particularly valuable in SE, 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 [3].

3 Illustrative Application Scenario

Before presenting CaLMo, in this section, we describe a scenario in which CLDs were used to support SE tasks. Our goal is to provide the reader with a real-world scenario in which CLDs were applied and use this scenario later as an example to demonstrate CaLMo's use. The scenario is detailed in 2022, which presents a case study carried out by the two last authors of this paper. 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 late and over budget, increasing 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. The researchers adopted a ST-based approach to identify the practices suitable for the organization and 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 [4]. 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.

[2]https://www.visual-paradigm.com/
[3]https://www.drawio.com/

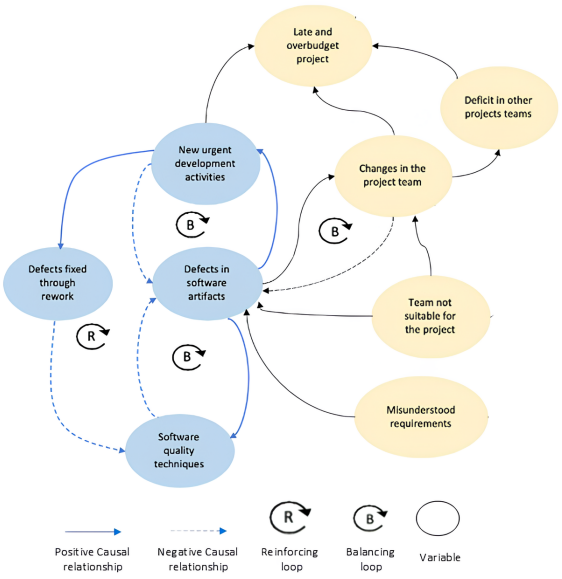

Figure 1: Fragment of CDL (adapted from [4])

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 using *Software quality techniques* that, when used, lead to fewer *Defects in software artifacts*.

The diagram in Figure 1 highlights in blue the *Shifting the Burden archetype*, including three *balancing feedback loops* and one *reinforcing feedback loop*. In this case, 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 reduce 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 *Software quality techniques*, which, when applied, help reduce defect recurrence in a more sustainable way. 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.

In the case study reported in 2022, the researchers created CLDs using general drawing tools (e.g., PowerPoint and Visio) and spent much effort creating and evolving CLDs together with the organization members. Each CLD was created separately, without integration with others. As a consequence, the consistency checking (e.g., ensuring that a variable has the same name in different CLDs) was made manually and was prone to errors.

4 CaLMo's Architecture and Structural Model

Figure 2 presents an overview of CaLMo's architecture. CaLMo [4] is a web-based tool designed to support the creation of CLDs. Targeting software engineers practitioners and researchers, CaLMo's full-stack implementation combines a front-end (using the *vis.js* library for dynamic graph visualization) with a modular back-end service, eliminating the need for specialized modeling expertise. User authentication was incorporated into the tool to enable personalized data storage, ensuring data privacy and isolation between accounts.

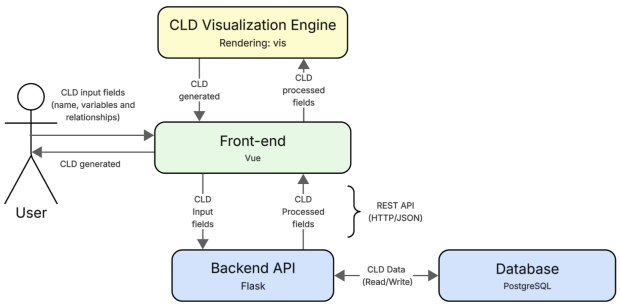

Figure 2: CaLMo architecture overview

We developed CaLMo's front end using *Vue.js* [5], a *JavaScript* framework based on a component-oriented structure and the *Model-View-ViewModel* (MVVM) architecture. This architectural pattern separates the user interface (View), the data and application logic (Model), and the intermediary layer (ViewModel) that binds them reactively.

For state management and navigation, we integrated *Pinia* and *Vue Router*, respectively. Both are the official state management and routing libraries that are recommended for *Vue* applications. To implement the interactive CLD visualization functionality, we used *vis.js*[6], a powerful data visualization library that enables dynamic node-edge graph rendering with interactive features such as node dragging, relationship visualization with colored arrows, and the ability to display causal polarity symbols (+ or -) on edges to indicate

[4]License: https://creativecommons.org/licenses/by/4.0/legalcode
[5]https://vuejs.org/
[6]https://visjs.org/

relationship types, enhancing the user experience when creating and analyzing CLDs

The back-end was implemented using *Flask*[7], a minimalist micro-framework written in *Python*. *Flask* enables rapid development while offering flexibility by allowing developers to include only the required libraries and extensions. The project was organized into distinct layers responsible for business logic, database interaction, and API endpoints. This modular structure enabled parallel development: one member of the team focused on the core back-end setup, while another implemented and adapted specialized algorithms for the identification and classification of feedback loops, as well as *archetypes*, ensuring seamless integration between both. The modular approach not only enhances scalability but also ensures maintainability, making the system easier to adapt and extend as the project grows.

For data persistence, we used *PostgreSQL*[8], a relational database system known for its reliability and ability to handle large volumes of information. For easier handling of queries, it was used in conjunction with *SQLAlchemy* ORM, a library that allows interactions with database records using Python objects instead of writing raw SQL, simplifying its use alongside the *Flask* framework. A key feature of this design is the many-to-many relationship between variables and CLDs, allowing users to reuse the same variables across multiple diagrams without duplication, promoting consistency, and enabling comparative analysis of different system representations that share common elements.

The Model component of CaLMo's architecture (*Model-View-ViewModel*) implements the structural model shown in the class diagram depicted in Figure 3. The model provides the underlying data structure for storing all CLD-related data, including variables, relationships (i.e., causal links), polarity, and archetypes. By formalizing these elements and their associations, the design ensures consistent persistence of the created CLDs and enables analytical features, such as the automatic detection of archetypes. In the model, there are three specific domain types, which are enumerated types, namely: *LoopType*, which can be Balancing or Reinforcing; *RelationshipType*, which can be Positive or Negative; and *ArchetypeType*, which can be: *Shifting the Burden*, *Fixes that Fail*, *Limits to Success*, *Success to the Successful*, *Tragedy of the Commons*, *Escalation*, *Growth and Underinvestment* or *Drifting Goals*.

5 CaLMo's Features

This section presents the core functionalities of CaLMo and describes, step by step, how users can build and analyze CLDs using the tool. We use the illustrative scenario presented in Section 3.

To start using CaLMo, the user must first access the tool through its web interface. New users are required to create an account to gain full access, enabling storage and management of their diagrams. Once logged in, the user is directed to the Dashboard homepage (Figure 4), which serves as the central navigation hub, offering immediate access to the core functionalities, which include: *create variables*, *create CLDs*, and *visualize CLDs*. On this page, the user can also access tutorial resources.

[7]https://flask.palletsprojects.com/en/stable/
[8]https://www.postgresql.org/

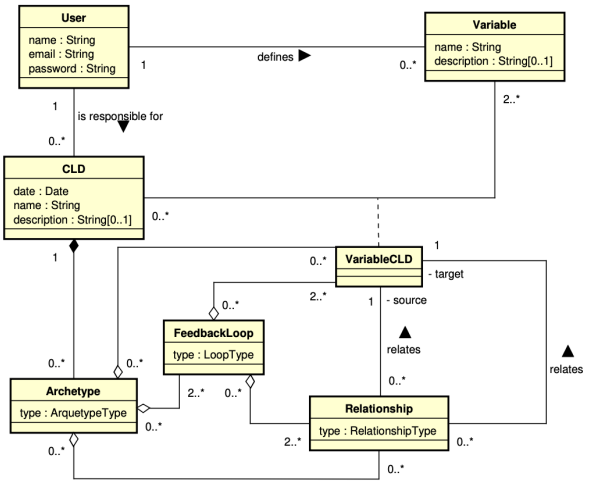

Figure 3: CaLMo's class diagram

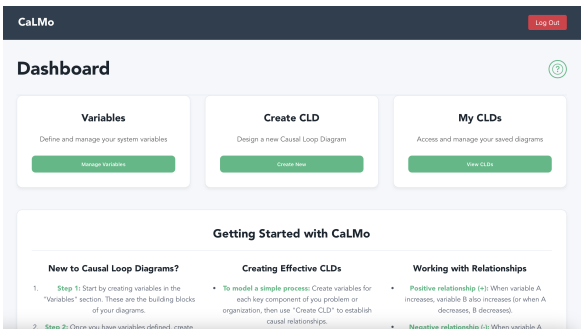

Figure 4: CaLMo's Homepage

To create a CLD, the user must first create variables to be included in the CLD. For that, the user selects the *Variables* option on the homepage and gets to the variables page (Figure 5), where they can record variables that will be later used in CLDs. Each variable must have a name and can have a description to enhance the model's clarity and maintainability. Following the scenario described in Section 3, Figure 1 illustrates some created variables and the addition of "*Software quality techniques*". By recording the variables, the user can add them to several CLDs. In this way, the tool reduces rework and helps keep consistency in different diagrams.

Following the creation of variables, the user can create CLDs using the recorded variables. In the *Create CLD* page (Figure 6) – accessed from the homepage – the user must identify the CLD to be created by providing its name and a description (optional) to help distinguish between multiple CLDs. The tool also records the CLD creation date. The user must, thus, select the variables to be added to the model and identify the relationships between them by indicating the source and target variables and the type of relationship (positive or negative). CaLMo checks the consistency of the informed relationships and prevents the user from making mistakes (e.g., contradictory polarities or circular references between variables). For example, if a relationship *R1* connects the

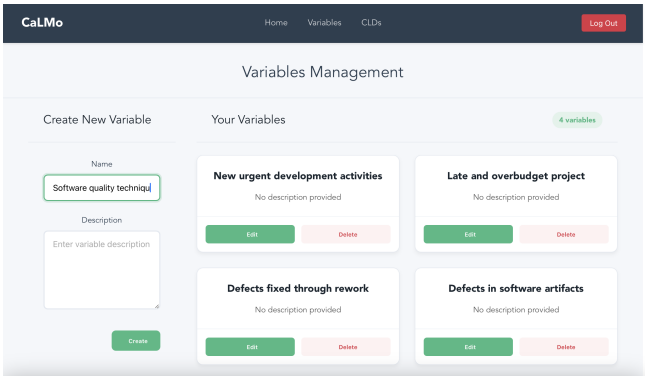

Figure 5: Variables

variables *a* (source) and *b* (target), and *R1* is a *positive* relationship, then there cannot exist a *negative* relationship *R2* connecting the variables *a* (source) and *b* (target) in the same CLD.

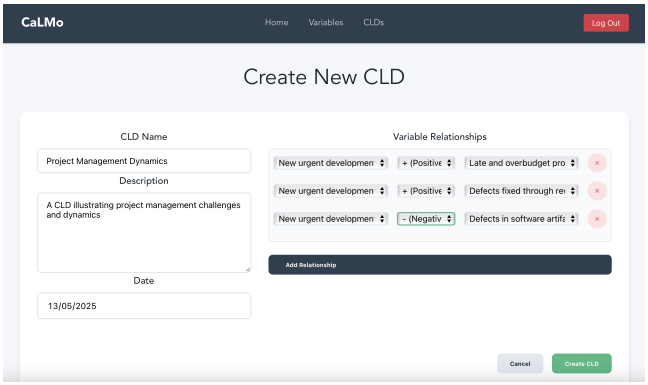

Figure 6: Establishing relationships between variables

Once the user informs the desired variables and relationships, the CLD is stored in the user's CLD Library and can be visualized. CaLMo automatically generates the CLD and presents it to the user (Figure 7). In the CLD, CaLMo automatically detects archetypes and presents them in a different color – currently, it supports the identification of the *Shifting the Burden* archetype. Figure 7 shows the CLD generated for the illustrative scenario (see Section 3). In the model, blue is used to highlight the "*Shifting the Burden*" archetype. The other variables appear in yellow. Positive relationships are represented in green, while the negative ones in red. Users can customize the CLD by repositioning nodes to improve layout clarity (all the changes made by the user in the CLD are saved).

CaLMo has analytical capabilities that extend beyond visualization. By clicking on variables, loops, or archetypes, the user accesses details about them (Figure 8), 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).

All created variables, relationships, and CLDs persist in CaLMo's repository (Figure 9), which organizes CLDs for easier access and

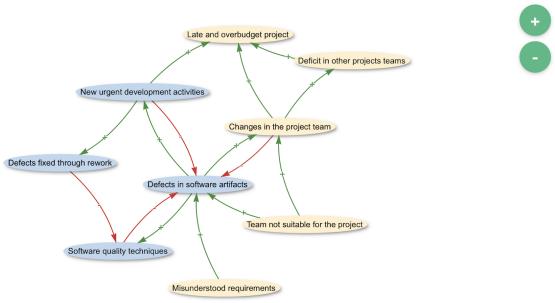

Figure 7: Generated CLD

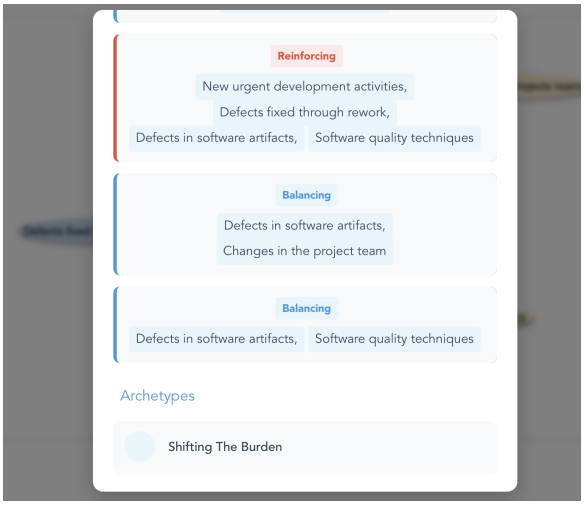

Figure 8: Feedback loop and archetype details

management. The interface supports editing and deleting to facilitate ongoing analysis.

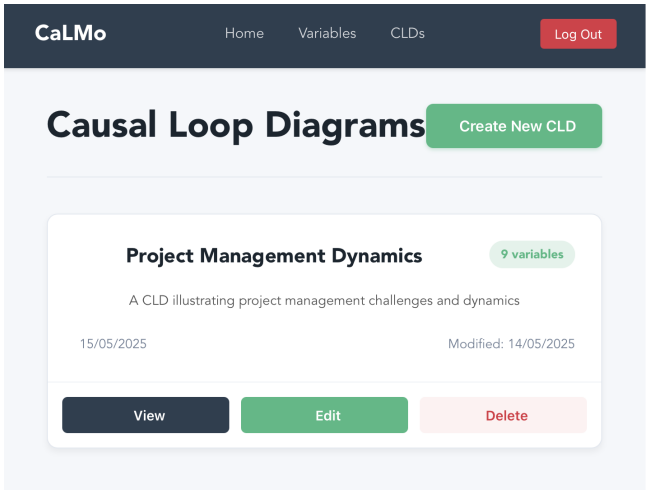

Figure 9: Example CLD Library

A new user can benefit from guidance through the Help Page (Figure 10), which combines conceptual explanations of CLD elements with a step-by-step guidance. The resource covers fundamental concepts, including variable definition, relationship polarity, archetype recognition, and feedback loop analysis, supporting the user during the modeling process, especially for those who are using the tool for the first time or are not very familiar with CLDs.

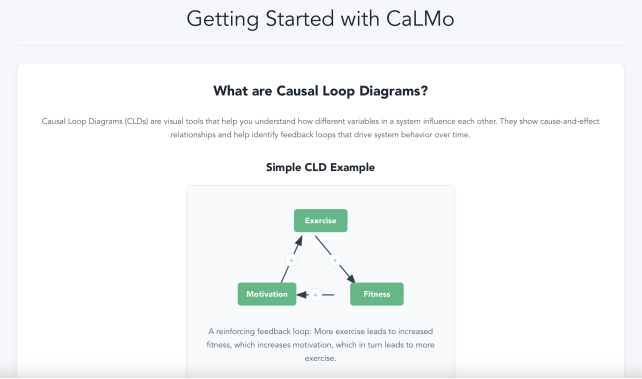

Figure 10: Help Page with CLD background and guidance

6 Implications

As noted in [4], one of the main challenges faced in the case study was the development of CLDs, which required knowledge of the ST modeling tool and demanded effort to create the CLDs without the support of a proper computational tool. We also identified this limitation in other studies [2]. CaLMo addresses this challenge by providing an interface and guidance that helps users create CLDs, even if they are novices in ST. CaLMo allows users to define variables and establish causal links with visual support. The tool also helps prevent modeling mistakes by enforcing constraints – for example, it prohibits self-referential variables and assigning multiple polarities to a single causal link.

Another advantage of CaLMo is its capability of automatically generating CLDs from the informed variables and relationships. As shown in Figure 7, the resulting diagram closely mirrors the "manually" constructed CLD presented in Figure 1. This automation not only streamlines the modeling process but also reduces the risk of human error, such as conflicting links or inconsistent labeling. Moreover, given that the tool automatically generates the CLD, the user can analyze the resulting CLD and, if changes are necessary, the user can easily change data (i.e., variables and relationships) and CaLMo generates a new CLD, demanding less effort than changing a diagram in general diagramming tools. Therefore, the user can create "tentative" or partial CLDs in an iterative modeling process until they reach the desired representation.

CaLMo also enables users to identify reinforcing and balancing feedback loops by directly interacting with the visualized nodes. These loops appear marked in the CLD, which facilitates the understanding of the system's dynamics. Furthermore, the tool identifies archetypes, providing users with insight into system behavior, without requiring deep knowledge or expertise to identify such patterns.

A distinguishing feature of CaLMo is the separation between variable creation and CLD construction. Variables are defined independently and, thus, can be reused across multiple diagrams. This approach supports consistency and scalability in modeling, as users can maintain a unified set of system elements while exploring different causal structures or scenarios. Additionally, because variables are managed outside individual CLDs, any updates to a variable's name or description are automatically reflected across all diagrams in which it appears, reducing redundancy, rework, and minimizing the risk of inconsistencies during iterative modeling processes.

Lastly, it should be noted that although we have developed CaLMo for the SE domain, it can be used to model systemic structures and behaviors across diverse fields.

7 Final Considerations

This paper introduced CaLMo, a tool to support the use of CLDs in SE. Its development was motivated by the promising use of ST in SE, the need to bridge the gap between theoretical potential and practical implementation [2], and by the practical experience of some of these authors, which demonstrated the value of ST in SE and revealed the knowledge and operational challenges of using it.

CaLMo helps create and analyze CLDs by offering a simple interface and guidance on CLD concepts and steps to develop it. CaLMo 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. Moreover, it automatically detects loops and archetypes and preserves semantic relationships for refinement. We found only one tool specific for CLDs [11] and, different from CalMo, it focuses on the visualization of CLDs generated from pre-built XMILE models instead of guiding CLDs creation.

Given the potential of ST in SE, in future work, we intend to build an ecosystem to support the use of several ST modeling tools in SE. CaLMo is the initial step in this path, and some improvements are planned. One of them has been addressed in an ongoing work to expand the archetypes automatically detected by CaLMo. Currently, it detects the *Shifting the Burden* archetype, and we are working on others to provide a more robust analytical capability. We also plan to carry out studies (case studies and experiments) to evaluate CaLMo use by other software engineers.

ARTIFACT AVAILABILITY

CaLMo can be run locally using Docker and Docker Compose. After cloning the repository from https://zenodo.org/records/15476020, users launch the back-end services (database and API) via docker-compose up, then start the front end from the frontend directory using *npm install*, followed by *npm run dev*. The application runs at localhost:3000. Currently, we are working on a web-hosted version to facilitate access to the tool.

ACKNOWLEDGMENTS

This research is supported by CAPES (Finance Code 001) and FAPES (Processes 2023-5L1FC, 2022-NGKM5, 2021-GL60J, and T.O. 1022/2022).

REFERENCES

[1] Monalessa Perini Barcellos. 2020. Towards a Framework for Continuous Software Engineering. In *Proceedings of the 34th Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 626–631. https://doi.org/10.1145/3422392.3422469

[2] Júlia Borges, Thiago Lahass, Amanda Apolinário, Paulo Santos Júnior, and Monalessa Barcellos. 2024. Unveiling the Landscape of System Thinking Modeling Tools Use in Software Engineering. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 47–57. https://doi.org/10.5753/sbes.2024.3232

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

[4] Paulo Sérgio dos Santos Júnior, Monalessa Perini Barcellos, and Rodrigo Fernandes Calhau. 2022. First step climbing the Stairway to Heaven Model-Results from a Case Study in Industry. *Journal of Software Engineering Research and Development* 10 (2022), 5–1. https://doi.org/10.5753/jserd.2021.1992

[5] Manfred Drack and Wilfried Apfalter. 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 (2007), 537–546. https://doi.org/10.1002/sres.855

[6] M. Fakhimi, D. Robertson, and T. Boness. 2021. THE BASIC PRINCIPLES OF SYSTEMS THINKING AND SYSTEM DYNAMICS. *Proceedings of the Operational Research Society Simulation Workshop* (2021). https://doi.org/10.36819/SW21.003

[7] ISACA. 2024. CMMI Model Quick Reference Guide: An Overview of the Capability Maturity Model Integration (CMMI)® Model. https://www.isaca.org/resources/reference-guide/cmmi-model-quick-reference-guide.

[8] P. John Sterman. 2000. *Business Dynamics: Systems Thinking and Modeling for a Complex World with CD-ROM.* McGraw-Hill Education.

[9] Daniel H Kim and Virginia Anderson. 1998. Systems archetype basics: From story to structure. Waltham: Pegasus Communications.

[10] Donella H Meadows. 2008. *Thinking in systems: A primer.* chelsea green publishing.

[11] William Schoenberg. 2020. LoopX: Visualizing and understanding the origins of dynamic model behavior. (2020). https://doi.org/10.48550/arXiv.1909.01138

[12] J. Sterman. 2010. *Business Dynamics: Systems Thinking And Modeling For The Complex World.* Tata McGraw Hill Education Private Limited. https://doi.org/10.1057/palgrave.jors.2601336