# Hardware-Driven Debt Accumulation: Classification and Early Indicators within Cyber-Physical Production Systems

L. Romier, B. Vogel-Heuser, Fellow, *IEEE*, L. Steuter

***Abstract*— In Cyber-Physical Production Systems (CPPS), Technical and Process Debt represent the long-term costs of suboptimal development decisions. As these systems grow in complexity, the lack of cross-disciplinary debt detection leads to debt accumulation, causing significant system instability. This paper presents a conceptual approach for early debt detection based on the analysis of five completed industrial projects. These were selected due to their cross-disciplinary complexity and hardware-centric constraints, ensuring a range of debt types within CPPS. First, a preliminary debt taxonomy, derived from these projects' analysis, is established to categorize the encountered debt into common subtypes. Second, influences and resulting debt states are derived, illustrating causal chains that lead to system-wide fragility. These connections are then quantified through a matrix that maps debt occurences and their influences based on the analyzed industrial projects. Finally, based on the insights gained, an initial set of heuristic-based indicators is synthesized to provide early visibility into development process health.**

## I. Introduction to Technical and Process Debt in Cyber-Physical Production Systems

Cyber-Physical Production Systems (CPPS) have gained significant attention due to their capability for smart manufacturing, allowing the control of physical processes and integration of enterprise resource planning on the shop floor [1]. However, as the complexity of these systems and their development process increases, the risk of Technical Debt (TD) accumulation rises. Originally used to describe the long-term cost of suboptimal coding decisions, TD serves as a metaphor for the trade-off between short-term delivery and long-term system health [2]. However, most TD research focuses on software engineering. While software-based TD mostly concerns code quality, architectural decisions, and documentation quality, CPPS are characterized by their unique architecture, including long lifecycles and tight interdependencies between software, hardware, and mechanical components. Current solutions lack cross-disciplinary analysis methods. As a result debt accumulates in hardware-near development processes that traditional software metrics fail to capture. There is thus a need for cross-disciplinary models that cover the different engineering disciplines involved, such as pure software engineering and industrial automation [3]. Process Debt (PD) refers to the debt accumulation resulting from suboptimal or outdated development practices that provide a temporary speed boost at the cost of long-term organizational stability. Within CPPS context, where software and hardware development must be synchronized, PD creates asymmetries that prevent the resolution of technical issues. This non-technical type of debt is difficult to detect in engineering artifacts such as code or diagrams, yet significantly increases future maintenance costs if left unaddressed [4].

To address these challenges, this paper presents the first proposal of a framework for early debt detection in industrial automation. First, a taxonomy of TD and PD specifically tailored to CPPS is established by analyzing historical data from five completed industrial projects. These projects, related to real-time control logic development and process lifecycle management, were selected for their cross-disciplinary complexity and reliance on synchronization between automation and software engineers. Building on this classification, the relations between TD/PD influences and resulting system states within CPPS context are mapped. To facilitate early detection, indicators that signal debt accumulation before system failure are derived. This Paper therefore extends TD and PD models by combining software-centric debt models with the cross-disciplinary, hardware-centric requirements of industrial automation. Specifically, it contributes (i) a first proposal of a debt classification taxonomy within the CPPS context using existing metrics, (ii) a mapping of causal chains between debt influences and resulting system states, and (iii) the identification of initial early debt indicators derived from recurring patterns observed across the analyzed projects.

The remainder of the paper is structured as follows. First, existing methods for debt assessment, detection, and repayment are reviewed in the state of the art. Next, a concept for the taxonomy of TD and PD in CPPS is provided, analyzing debt cases and introducing causal chains linking influences to resulting states. Then, early indicators are derived from the taxonomy proposal, allowing early TD and PD detection within CPPS development. Finally, the paper's findings are reviewed, and future requirements for debt management are discussed.

## II. State of the Art regarding Technical and Process Debt in CPPS

TD in mechatronics often originates in early lifecycle stages with cross-disciplinary effects, showing how management decisions affect the downstream development process [5]. A framework for identifying TD within CPPS shows how inappropriate tools and knowledge asymmetry create debt [6]. However, these approaches neglect hardware-driven constraints characterizing cross-disciplinary CPPS development. An automated TD identification approach by Avgeriou et al. integrates versioning systems as data sources [7], while a research review advocates a shift from code-based analysis to architectural TD management by integrating technical tools with financial models [8]. While providing a future roadmap, these are mostly software-centric and rely on versioning data that overlooks the automation-related

All authors are with the Technical University of Munich, Garching, Germany, at the Institute of Automation and Information Systems (e-mail: {lucas.romier, vogel-heuser, luis.steuter}@tum.de).

cross-dependence within CPPS. Similarly, Martini et al. propose a risk-based prioritization framework for TD repayment in a microservice-based architecture [9]. Combining TD and PD, another study examines how TD is psychologically taxing and negatively influences subsequent development [10]. These frameworks are tailored to software environments or microservice and lack the cross-disciplinary approach required for CPPS. A qualitative case study on agile environments identifies five key categories (social dynamics, process, people, documentation, and requirements) where knowledge asymmetry and weak governance directly generate TD accumulation [11]. Another framework introduces TD assessment and prioritization for digital twin development, unifying simulation and software engineering [12]. While highlighting debt causes, these do not assess TD and PD from hardware-driven CPPS development. An industrial case study reveals that TD alone fails to fully explain long resolution times, as task complexity and organizational constraints show significant influence [13]. Finally, a machine learning approach aggregates data from multiple analysis tools, demonstrating that multiple classifiers can identify TD more reliably than single-tool identification methods [14]. These approaches focus on issue resolution and tool-based analysis, but lack a cross-disciplinary approach to detect early signals of debt in CPPS.

## III. Concept for the Taxonomy of Hardware-Driven Technical and Process Debt in CPPS

As manufacturing systems become more complex, shortcuts taken during development accumulate as debt, reducing system stability and increasing maintenance effort. Formalizing the relationship between technical shortcuts and organizational inefficiencies helps identify early signs of instability before major rework or malfunctions occur. The empirical basis of this research is a qualitative analysis of five industry cases in the automotive and manufacturing automation. Data was collected through semi-structured interviews with two developers specializing in automation engineering. Cases were purposively selected from projects that required significant post-completion rework. This paper identifies how specific development decisions translate into long-term operational risks in CPPS.

### *A. A taxonomy of TD and PD in CPPS*

The taxonomy in Table I is derived from previously established frameworks by Vogel-Heuser et al. [3, 5] and Avgeriou et al. [2], whereas the distinction between TD and PD follows the work of Ahmad et al. [11]. It was then further refined through classified industry projects in real-time control and lifecycle management, serving as a collection of documented industry examples. To capture these instances, semi-structured interviews were conducted with two independent experts, focusing on identifying TD and PD occurrences encountered

Table I - Taxonomy of Technical and Process Debt in CPPS

| | TD/PD Type | CPPS Stage | Typical Causes | Long-Term Impact |
|---|---|---|---|---|
| TD | Architecture TD | Software Development | Purely AI-Generated Code | Faulty/Inconsistent Architectures |
| | | | Vendor-Driven Architectures | No System Portability |
| | Design TD | Software Development | Undocumented Design Choices | Incorrect Design Extensions |
| | | | Purely AI-Generated Code | Faulty/Bad Code |
| | | Hardware Integration | Missing Real-Time Specifications | Performance Degradation under Load |
| | | | Coupling of Control Logic and I/O | No System Portability |
| | Test TD | Company Organization | Only Manual Testing | Lack of Transparency |
| | | Software Development | Missing/Incomplete Test Cases | Unassessed System Correctness |
| | Infrastructure TD | Company Organization | Decisions without Lifecycle Planning | Early System Obsolescence |
| | | | Different Platforms across Locations | System Incompatibility |
| | | Hardware Integration | Hardware Retrofitting due to Cost/Availability | Limited System Observability |
| | | | Late I/O Integration without Design | No System Scalability |
| | Versioning TD | Company Organization | Missing Version Control | Diverging Versions Over Time |
| | | | Lack of Company Libraries | Redundant Implementations |
| PD | Role PD | Company Organization | Unclear System Ownership | Delayed/Inefficient Decision-Making |
| | Requirements PD | Company Organization | Undocumented Requirements | Diff. Implementation ↔ Expected Behavior |
| | | | No Requirement Sync. across Locs. | Diff. Implementation ↔ Expected Behavior |
| | Documentation PD | Company Organization | Documentation is treated as Secondary | Strong dependency on individual experts |
| | | | Missing Handover Procedures | Loss of Project Knowledge |
| | | | Missing Documentation Standards | Increased Induction Period |
| | Synchronization PD | Company Organization | Knowledge Diffs. across Locations | Dependency on Expert Location |
| | | | Unsynchronized Changes across Locations | Difficult Fault Localization |
| | | | Insufficient Comm. across Locations | Diff. Implementation ↔ Expected Behavior |
| | | | Separation between IT, OT, and Maintenance | Knowledge Silos inside Departments |
| | | Hardware Integration | No Process for Firmware/Hardware Updates | Escalating Maintenance Costs |
| | Unsuitable Process PD | Company Organization | Processes not Suited for Distributed Dev. | Inefficient Coordination across Locations |
| | | | No Governance for AI-Assisted Development | Uncontrolled Codebase |
| | Infrastructure PD | Company Organization | Fragmented Tooling Strat. across Locations | System Incompatibility |
| | | | Local Infrastructure Decisions | No Organization-Wide Standardization |

after project completion. Following identification, collaborative sessions were held to discuss the problems responsible for the accumulated debt and reconstruct its possible causes. The introduced table's columns therefore display first-hand accounts of the causes and results identified during these discussions. The first column of the taxonomy table, **TD/PD Type**, distinguishes Technical Debt, mainly found in engineering artifacts, from Process Debt, which describes organizational shortcomings. The **CPPS Stage** column indicates which stage or category of the hardware-near development process the debt falls under. Here, the *Company Organization* stage describes shortcomings in strategic and management practices, focusing on how, e.g., missing governance or unclear ownership can lead to debt accumulation. Next, the *Software Development* stage describes hardware-driven software design and implementation, where debt often manifests as, e.g., eroded architecture or fragmented infrastructure. The *Hardware Integration* stage is linked to operational and hardware aspects of the system, where debt results in, e.g., limited observability or the need for short-term hardware retrofitting after deployment. Lastly, the **Typical Causes** column briefly describes the specific events or decisions that triggered the debt, while the **Long-Term Impact** column specifies its long-term consequences.

The following debt types were detected across the completed projects. Each category is based on examples of encountered debt within the projects, providing a basis for the following debt descriptions. **Architecture and Design TD** were mostly linked to poor software architecture choices, often driven by purely AI-generated code or vendor-driven constraints. **Versioning and Test TD** mainly occurred because system stability was ignored during the software development process, often due to reliance on manual testing or the absence of version control, leading to unassessed system correctness and divergent software variants. **Infrastructure TD** was primarily encountered due to hardware retrofitting or late hardware-to-software integration, limiting the system's observability and preventing future scalability. **Role and Requirements PD** predominantly occurred due to unclear system ownership and undocumented requirements during development, resulting in an asymmetry between the system's expected behavior and actual capabilities. **Documentation and Synchronization PD** often further reinforced these issues when changes were not synchronized across varied company locations, leading to knowledge silos or increased reliance on external experts. Finally, **Unsuitable Process and Infrastructure PD** address the governance strategy of the development process itself, where a lack of oversight of AI-assisted tools or fragmented tooling strategies often leads to an uncontrolled codebase and a lack of standardization within the organization.

### *B. The Influences and Propagation of TD and PD in CPPS*

While the introduced taxonomy categorizes individual debt types across industrial examples, it does not show the interconnected nature of the accumulated debt. Therefore, Figure 1 is constructed to illustrate the causal chain through which debt manifests as additional influences or resulting instability. To establish the causal chains, a cross-case analysis was conducted, mapping and finding similarities within the previously identified debt and its causes to reveal recurring patterns of debt accumulation. In the provided graph, influences represent the root causes that trigger debt accumulation. These serve as starting points of the causal chain and can be categorized into three primary categories. This categorization was based on the analysis of interview data on overlaps and similarities across projects, highlighting the systemic nature of debt in cross-disciplinary environments. The first group, **Organizational and Process Misalignment**, including *Missing Location Synchronization*, *Requirements Misalignment*, *Unclear System Ownership*, and *Unsuitable Development Processes*. These influences reflect a lack of coordination or poorly defined responsibilities. The second group relates to **Knowledge and Governance Gaps**, where *Location-Based Knowledge Gaps*, *Low Documentation Maturity*, and specific governance gaps, such as *Missing AI Governance* and *Missing Lifecycle Governance*, force engineers to make individual decisions for want of necessary information and guidelines. A third group focuses on **Technical and Strategic Constraints**, including *Fragmented Tooling and Infrastructure*, *Vendor-Driven Architectures*, and *Delayed / Local Decision Making*. Even actions taken to resolve existing issues, such as *Ad-Hoc Architectural Decisions* and *Reactive Fixes*, are categorized as influences, as they often introduce new compromises instead of long-term solutions. It is noteworthy that the debt cycle is not linear. Any influence state can serve as a starting point for debt, regardless of whether it originated in a previous step in the chain. The non-linear nature of these cycles was systematically validated through a series of interviews with the two experts, both lead engineers from the studied CPPS

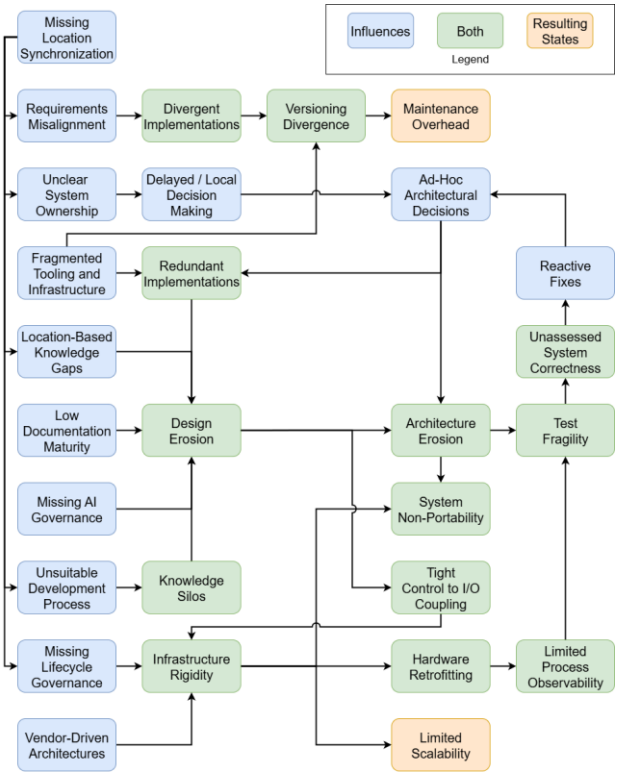

Figure 1 - Influences and Propagation of TD and PD

projects. They confirmed that reactive measures to mitigate debt often led to further debt accumulation. Based on this feedback, the model shifted from a simple causal chain to a dynamic system, where resulting instabilities often become new influences for further reactive decision-making. The experts also confirmed that debt propagation is influenced by the cross-disciplinary nature of CPPS projects and the asymmetry of refactoring. While software-based debt can theoretically be refactored, physically based debt, such as *Tight Control to I/O Coupling* or *Hardware Retrofitting*, introduces mechanical constraints that are difficult to mitigate. This creates a hierarchical dependency in which the software cannot be optimized without expensive mechanical redesign, forcing reactive adaptation of software architectures. For instance, a suboptimal physical sensor that requires additional filtering logic in the code forces the embedding of physical constraints into the software development process, leading to inter-disciplinary debt accumulation.

The introduced influences then lead to states that show the system's instabilities and the long-term impact of accumulated debt, such as *Maintenance Overhead* and *Limited Scalability*. At this stage, the debt has matured into a significant burden, reducing system performance and the engineers' ability to implement future changes efficiently. Between the initial causes and resulting states are states that can be categorized as both. These serve as a bridge in the causal chain, acting simultaneously as effects of previous debt and as new influences that introduce further issues. Therefore, they serve both as starting points for debt and, like resulting states, have a direct negative impact on the system's stability. The first group, **Architectural and Functional Decay**, describes system erosion through states such as *Design Erosion*, *Architecture Erosion*, *and Knowledge Silos,* leading to *Divergent*, *Versioning*, or *Redundant Implementations*. The second group, **Systemic Rigidity and Quality Gaps**, describes the loss of system flexibility and process oversight through *Infrastructure Rigidity*, *System Non-Portability*, *Tight Control to I/O Coupling*, *Hardware Retrofitting*, *Unassessed System Correctness*, *Test Fragility*, and *Limited Process Observability*. By acting as both influences and outcomes, these states show that TD is not a singular event. TD and PD often feed a self-sustaining cycle of influences and more accumulated debt.

### *C. The cross-case mapping of TD and PD*

While the introduced causal graph emphasizes the most prominent causal debt chains, it is limited by its focus on only the most prominent interactions. In complex CPPS environments, the causal chains linking many influences to their resulting states remain hidden, acting on the system without an obvious direct link. Therefore, to ensure analytical completeness, we transition from a static graph to the Matrix presented in Table II, which maps all influences to all resulting states. Thus, the goal is to map the interactions identified in the causal chain into a structured, quantifiable format, enabling clearer analysis of how influences lead to system-wide instabilities. The matrix columns represent the influences, including both original root causes and intermediate states that trigger additional debt. Moreover, related influences have been combined into broader categories. For instance, *System Erosion* combines Design and Architecture Erosion, while *Missing Governance* unites Missing Lifecycle Governance and Missing AI Governance. *System Divergence* combines Divergent Implementations and Versioning Divergence, and *Flexibility Loss* groups System Non-Portability and Infrastructure Rigidity. Furthermore, *Hardware Dependency* combines Tight Control-to-I/O Coupling and Hardware Retrofitting, and *Unassessed Stability* combines Unassessed System Correctness, Limited Process Observability, and Test Fragility. *Reactive Adjustments* include both Ad-Hoc Architectural Decisions and Reactive Fixes, while *Lacking Processes* unite Unsuitable Development Processes, Requirements Misalignment, and Fragmented Tooling and Infrastructure. Standalone influences such as Knowledge Asymmetry, Low Documentation Maturity, and Vendor Dependency are also included as individual columns.

The matrix rows represent the resulting states, including both final instabilities and intermediate states. The matrix was populated using the following method: First, the columns and rows were derived from the previously established causal graph. To ensure an unbiased assessment, the matrix was provided to the two previously interviewed experts as an empty matrix. They were then tasked with determining whether a link between a specific influence and a resulting state existed based on their observations. Next, for each case, the causal chain between influences and resulting states was analyzed and summarized as collective findings. This systematic process transformed qualitative interview data into quantifiable distribution data, enabling case-specific filtering of debt accumulation and its underlying influences. To represent the influences, symbolic notation was used within the matrix cells. An empty circle indicates that a link between an influence and a state was denoted in none or only one project, suggesting a case-specific occurrence. A half-filled circle indicates that a connection was

Table II - Cross-Case Patterns of TD and PD

| Resulting States | Knowledge Asymmetry | Low Documentation Maturity | System Erosion | System Divergence | Flexibility Loss | Missing Governance | Hardware Dependency | Reactive Adjustments | Unassessed Stability | Lacking Processes | Vendor Dependency |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Grouped Influences** | | | | | | | | | | |
| Design Erosion | ● | ● | ◐ | ◐ | ◐ | ● | ○ | ● | ○ | ● | ○ |
| Architecture Erosion | ● | ● | ◐ | ◐ | ● | ● | ○ | ● | ◐ | ● | ○ |
| Versioning Divergence | ● | ● | ○ | ◐ | ○ | ● | ○ | ◐ | ○ | ● | ○ |
| Divergent Implementations | ● | ● | ◐ | ◐ | ◐ | ● | ◐ | ● | ◐ | ● | ◐ |
| Redundant Implementations | ● | ● | ● | ● | ◐ | ● | ○ | ● | ◐ | ● | ○ |
| Knowledge Silos | ● | ● | ◐ | ◐ | ○ | ● | ○ | ○ | ○ | ● | ○ |
| Infrastructure Rigidity | ● | ● | ◐ | ◐ | ● | ● | ◐ | ◐ | ◐ | ● | ◐ |
| Tight Control-to-I/O Coupling | ○ | ○ | ◐ | ○ | ◐ | ◐ | ◐ | ◐ | ○ | ◐ | ○ |
| System Non-Portability | ● | ● | ● | ● | ● | ◐ | ◐ | ◐ | ● | ● | ◐ |
| Limited Scalability | ● | ● | ● | ● | ● | ◐ | ◐ | ● | ● | ● | ◐ |
| Hardware Retrofitting | ○ | ○ | ◐ | ○ | ○ | ◐ | ○ | ◐ | ○ | ○ | ○ |
| Limited Process Observability | ● | ● | ● | ● | ◐ | ● | ○ | ● | ● | ● | ○ |
| Test Fragility | ● | ● | ● | ● | ◐ | ● | ◐ | ● | ● | ● | ○ |
| Unassessed System Correctness | ● | ● | ● | ● | ◐ | ● | ◐ | ● | ● | ● | ○ |

*Note: To read the matrix, a row state results from column influences, with the intersection indicating the encountered frequency across all analyzed projects. For example, tracing the "Knowledge Asymmetry" column to the "Architecture Erosion" row shows that most projects reported lacking shared knowledge as a direct driver of structural decay in system architecture.*

observed in two or three of the analyzed projects, suggesting an already recurrent, but still context-dependent pattern. Finally, a full circle indicates that the relationship was denoted in four or five out of the five total projects, representing a systemic and consistent debt accumulation path.

The matrix reveals several cells where specific influences consistently led to TD states across most projects. Among the influences, Knowledge Asymmetry, Low Documentation Maturity, Missing Governance, and Lacking Process were most responsible for the resulting states, each acting as a significant influence. Similarly, among the resulting states, Divergent Implementations, Infrastructure Rigidity, System Non-Portability, Limited Scalability, Limited Process Observability, Test Fragility, and Unassessed System Correctness occurred due to almost all influences. This indicates that these states occur in most projects under stress. Hardware-near influences, such as Hardware Dependency, mostly lead to specific issues, such as System Non-Portability, but do not carry the same weight as process-related influences, which influenced the further debt accumulation across all analyzed cases.

Mapping also reveals self-reinforcing cycles, where states and influences reinforce one another, creating a downward spiral of accumulated TD. First, the cycle between Design Erosion and System Divergence is noticeable. As system maintainability decreases, developers resort to Divergent or Redundant Implementations to meet immediate needs, which in turn results in System Erosion and Infrastructure Rigidity. Similarly, a cycle between Reactive Adjustments, Lacking Processes, and Test Fragility is noticeable. When improvised fixes are applied, test cases often become unreliable or are omitted. This then leads to Limited Process Observability, where observed system stability remains unknown, often forcing further improvised fixes.

## IV. Derived Early Indicators of Technical and Process Debt in CPPS

To avoid the resulting states and self-reinforcing loops identified previously, shifting from retrospective analysis to active detection is essential to managing and avoiding TD. When issues are identified only after they occur, system- and process-rooted instabilities often prevent cost-effective mitigation, as the accumulated debt is too large to repay within the standard development process. To prevent these states from taking hold, observable and early-stage symptoms must be identified during the development and maintenance phases. Using these indicators allows early intervention, thereby ensuring the system's stability. These indicators are detected through continuous monitoring that regularly audits process artifacts. By identifying localized engineering artifacts or architectural anomalies, teams can identify debt accumulation at its earliest stage before it manifests as, e.g., physical system failures or unmaintainable codebases.

Table III summarizes these observable indicators synthesized from the completed projects, identified TD influences, and their propagation. These indicators are based on a secondary analysis of previously collected project data, focusing on identifying early observable signals of debt states. These findings were then reviewed in a new round of developer feedback, confirming the reliability of the observable symptoms. The proposed indicators are categorized by debt type: TD, PD, or a combination of both. The table highlights how specific behaviors, such as prioritizing short-term cost savings over maintainability or the absence of peer reviews, act as early warnings for the organizational and TD states discussed earlier. Classifying indicators as both TD and PD highlights the cross-disciplinarity of CPPS development. In these cases, a physical constraint or decision serves as a rigid anchor, influencing the development process, or vice versa. Notably, these hybrid indicators describe scenarios in which a hardware limitation, like legacy hardware usage, directly requires reactive software adaptation to maintain operationality. Ultimately, these indicators give early visibility into development process health and, when used, limit debt accumulation.

## V. Conclusion

This paper addressed Technical Debt and Process Debt in the domain of Cyber-Physical Production Systems by analyzing five completed industrial projects to derive a taxonomy of debt cases. By combining software-centric debt analysis with the physical constraints of industrial automation and specific, cross-disciplinary process structures, the introduced taxonomy demonstrates that existing TD classifications from traditional software engineering and standard PD classifications are insufficient for the hardware-centric nature of CPPS development. Extensive mapping required specific extensions to address cross-domain dependencies and tight coupling at the control layer, successfully revealing critical causal links between development oversights and visible debt outcomes. Mapping these influences onto resulting states exposes causal chains where debt accumulates through intermediate states of architectural and functional system decay, transforming into self-reinforcing cycles that grow into a significant operational burden marked by high maintenance, limited scalability, and poor operability. Finally, early indicators of TD and PD were identified to facilitate proactive detection and enable well-documented, traceable decision-making, ensuring short-term gains do not prevent subsequent production system development.

These findings show that TD and PD in CPPS are not isolated incidents but often enter self-reinforcing cycles of further debt accumulation. While the analyzed cases represent only a small sample, they offer the qualitative depth needed to identify these critical patterns. However, increasing complexity in cross-disciplinary, hardware-centric CPPS development requires automated, tool-assisted strategies for early debt detection and repayment. Without a strategy to assess, prioritize, and repay debt, the technical and organizational debt states identified in this study will continue to prevent the scalability and usability of intelligent manufacturing systems. By applying these findings to

Table III - Observable Indicators of TD and PD

| TD/PD Case | Early Indicators | Debt Type |
|---|---|---|
| Missing Location Synchronization | Same PLC function block implemented differently in two plants | TD |
| | Bug fixed at Site A but still present at Site B months later | TD & PD |
| | Lack of synchronization meetings across sites | PD |
| | Location-specific product roadmaps with no shared milestones | |
| Unclear System Ownership | IT and OT teams each assume the other owns architectural decisions | PD |
| | Design changes approved without documentation or task assignment | |
| | Escalations circulate between departments without resolution | |
| Fragmented Tooling/Infrastructure | Different PLC vendors or IDE versions across locations | TD |
| | Limited tool or support availability due to region locks | |
| | Engineers maintain personal deployment scripts | TD & PD |
| Knowledge Asymmetry across Locations | Only one location can debug production issues | TD & PD |
| | Engineering artifacts stored locally at a single site | |
| | Cross-location reviews avoided due to knowledge gaps | PD |
| Knowledge Asymmetry within Locations | "Only X knows this subsystem" repeatedly stated | TD & PD |
| | Changes postponed due to absence of key person | PD |
| | No peer review for engineering artifacts | |
| Low Documentation Maturity | Engineering artifacts contradict current implementation | TD |
| | No record of or no central design decisions | PD |
| | New engineers require weeks of shadowing | TD & PD |
| Missing AI Governance | AI-generated PLC or control logic committed without disclosure | PD |
| | No agreed rules on acceptable AI usage | |
| | Code reviews focus on syntax, not architectural intent | TD & PD |
| Unsuitable Development Process | No code review or insufficient code review | PD |
| | Changes deployed directly to production systems | |
| | Missing engineering artifact traceability | |
| | No structured feedback from maintenance to development | |
| Missing Lifecycle Governance | Legacy hardware retained without replacement plan | TD & PD |
| | No scheduled refactoring or modernization phases | |
| | Short-term cost savings prioritized over maintainability | |
| Vendor-Driven Architectures | Architecture mirrors vendor demo examples | TD |
| | Interfaces tightly coupled to proprietary APIs | |
| | Vendor-specific customer requirements | |
| | Restrictions on use of alternative vendors | |
| | Vendor changes force system redesigns | |
| External Expert/Freelancer Dependency | External contractors required for routine changes | TD & PD |
| | Internal staff lack access to produced engineering artifacts | |
| | Knowledge transfer limited to brief handovers | PD |

navigate development trade-offs, practitioners can achieve traceable decision-making and system stability.

## REFERENCES


[1] X. Yao et al., "Smart manufacturing based on cyber-physical systems and beyond," Journal of Intelligent Manufacturing, vol. 30, Dec 2019.

[2] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," Journal of Systems and Software, vol. 101, pp. 193–220, 2015.

[3] B. Vogel-Heuser and S. Rösch, "Applicability of technical debt as a concept to understand obstacles for evolution of automated production systems," in 2015 IEEE International Conference on Systems, Man, and Cybernetics, 2015, pp. 127–132.

[4] A. Martini et al., "Process debt: Definition, risks, and management," Journal of Software: Evolution and Process, vol. 37, no. 4, 2025.

[5] B. Vogel-Heuser and F. Bi. "Interdisciplinary effects of technical debt in companies with mechatronic products – a qualitative study," Journal of Systems and Software, vol. 171, no. 1, pp. 1-17, Aug. 2020.

[6] L. Waltersdorfer, F. Rinker, L. Kathrein, and S. Biffl, "Experiences with technical debt and management strategies in production systems engineering," in 2020 IEEE/ACM International Conference on Technical Debt (TechDebt), 2020, pp. 41–50.

[7] Y. Li, M. Soliman, and P. Avgeriou, "Automatic identification of self-admitted technical debt from four different sources," Empirical Software Engineering, vol. 28, no. 3, p. 65, Apr. 2023.

[8] P. Avgeriou et al., "Technical debt management: The road ahead for successful software delivery," in 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE). IEEE, May 2023, p. 15–30.

[9] S. S. De Toledo et al., "Accumulation and prioritization of architectural debt in three companies migrating to microservices," IEEE Access, vol. 10, pp. 37 422–37 445, 2022.

[10] J. Olsson et al., "Measuring affective states from technical debt," Empirical Software Engineering, vol. 26, no. 5, p. 105, Jul. 2021.

[11] M. O. Ahmad et al., "Technical debt is not just technical: An industrial case study in large agile software development," Journal of Systems and Software, vol. 234, p. 112719, 2026.

[12] S. Malakuti, "Emerging technical debt in digital twin systems," in 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA ), 2021, pp. 01–04.

[13] B. Paudel et al., "Exploring the Relationship Between Technical Debt and Lead Time: An Industrial Case Study," in 2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Mar. 2025, pp. 693–703.

[14] D. Tsoukalas et al., "Machine learning for technical debt identification," IEEE Transactions on Software Engineering, vol. 48, no. 12, pp. 4892–4906, 2022.