IEEE Open Journal of the Systems Engineering

Received 10 June 2023; revised 2 September 2023; accepted 7 September 2023. Date of publication 18 September 2023; date of current version 23 October 2023.

Digital Object Identifier 10.1109/OJSE.2023.3316395

Ontology for Technical Debt in Systems Engineering

HOWARD KLEINWAKS, ANN BATCHELOR, AND THOMAS H. BRADLEY (Member, IEEE)

Department of Systems Engineering, Colorado State University, Fort Collins, CO 80523 USA

CORRESPONDING AUTHOR: HOWARD KLEINWAKS (e-mail: howard.kleinwaks@colostate.edu)

ABSTRACT The technical debt metaphor is used to describe the long-term consequences of engineering decisions made to achieve a short-term benefit. The metaphor originated in the field of software engineering and has begun to migrate to other fields, including systems engineering. The usage of the metaphor, its associated terminology, and basic definitions vary both within the software field and within the greater engineering community. The lack of consistent definitions inhibits the ability of system developers to understand and control technical debt within their system developments. This article presents an ontology for technical debt, focusing on the field of systems engineering. By providing a set of concise and consolidated definitions, this ontology enables precise discussion of technical debt and associated techniques for mitigating its impact within systems engineering.

INDEX TERMS Ontology, rework, systems engineering, technical debt (TD).

I. INTRODUCTION

Technical debt (TD), originally defined within the context of software engineering [1], is becoming a standard part of the technical lexicon, used by system engineers [2], program managers [3], and corporate executives [4]. But what exactly is TD? It has variously been defined as the long-term impact of compromises made for the short-term benefit [5], the difference between the planned system capabilities and the actual system capabilities [6], a promise to complete work in the future [7], the acceptance of a short-term solution that will create additional work in the long-term [2], and all the "technical work that has to be completed in the future" [4]. Further complicating the problem is the use of different terms, such as rework [8] and unintended consequences [9], to define similar problems. Even these terms do not have consistent definitions, as up to eight different definitions of rework have been found within the same paper [10]. These conflicting sources make it clear that a common definition of TD does not exist neither in the broader research community nor specifically within the field of systems engineering [11].

The definitions of the components of TD also vary from author to author. Tom et al. [12] mapped the components of TD to the associated forms of TD, showing that multiple components can be classified into more than one form of debt. Li et al. [5] defined the "cause" of TD as "the reason for the existence of technical debt," which corresponds to the "precedent" defined by Tom et al. [12]. Rios et al. [13] used the term "consequence" to identify the impacts of TD on the system, while Tom et al. [10] discussed the impacts in terms of the "attributes of technical debt." Alves et al. [14] defined an ontology of TD types but did not provide details on terminology beyond those types.

To address the terminology differences, several authors have developed taxonomies of TD. A taxonomy provides methods to classify items, while an ontology provides definitions of those items [15]. Taxonomies are necessary to enable the classification of different TD types; however, an accepted ontology is required to provide the basis for those taxonomies. Yang et al. [16] recently defined a taxonomy focused on the incorporation of custom off-the-shelf products into complex systems. They expand Kruchten's TD landscape [17] to include an additional "Accountability" access and define several factors leading to different types of TD. Tom et al. [12] also defined a taxonomy of TD, including methods for classifying the TD based on precedents, outcomes, and attributes. They defined several attributes of TD, such as technical bankruptcy, but did not provide a full ontology and their definitions did not necessarily extend beyond the field of software engineering. Alves et al. [14] extended their earlier work to provide a taxonomy of TD types [18]. Several

other authors have proposed taxonomies related to TD [13], [19], [20], but the taxonomies focus on classifying TD and do not provide consistent definitions that can be used across industries [21].

Furthering the problem, TD is not well-researched within systems engineering literature [22]. The authors have provided empirical evidence that TD does occur within systems engineering, even if the terminology is not widely used [23]. This survey identified that TD is more likely to be created early in the system lifecycle and its impacts are more likely to be observed later in the system lifecycle. The lack of a common ontology for common systems engineering problems prevents the systematic identification of similar research and, therefore, the sharing of tools and techniques to manage TD and mitigate its impact throughout the system lifecycle [21]. With TD occupying significant portions of corporate technology portfolios [4], the management of TD is increasing in importance and value. Within specific systems engineering contexts, the problem of TD is increasing with the push to release products on shorter timelines [24] and an increased emphasis on prioritizing value delivery over nonfunctional requirements [25]. These pressures can result in developers taking shortcuts [26] and systems that break more easily when changes are required and which are more difficult to maintain, both of which are symptoms of unpaid TD [27].

Based on this examination of the state of the art in the field, it is clear that there is a need for a common ontology for TD. While multiple taxonomies exist, the authors are unaware of a comprehensive ontology of TD, particularly when considered in the field of systems engineering. Establishing a common language for TD is a key step to enable cooperation between the business and technology sectors of a company [28] and to enable communication between practitioners throughout the field.

Communication between practitioners is especially important as systems become increasingly complex and combined into systems of systems. In these cases, TD can be incurred in one system and then compound throughout the systems of systems. With increased complexity, identifying the source of the problem so that it can be remedied can become difficult and expensive, especially since the source of the TD may be far removed from its impacts. These factors are exacerbated within systems engineering, compared to software engineering, due to the increased interactions with external influences that may be outside the control of the system developer. Therefore, an ontology that provides a common basis for discussions across the entire system context is critical to managing systematic risks.

This article develops such an ontology for the field of systems engineering, which will enable a consistent discussion about TD and its management within systems engineering [15]. Standardization of terminology and definitions will lead to knowledge sharing and the development of measurement and management techniques.

The rest of this article is structured as follows. Section II presents the proposed ontology for TD for systems engineering. Section III discusses the use of ontology. Section IV concludes the article and presents concepts for future work.

II. TECHNICAL DEBT ONTOLOGY FOR SYSTEMS ENGINEERING

A. TECHNICAL DEBT CONCEPT MAP

The development of an ontology for TD starts with a conceptual understanding of TD. The following example, originally provided in a survey on the prevalence of TD within systems engineering [23], demonstrates how TD can impact the development of a system.

> Sydney is a test engineer tasked with writing test procedures to ensure that each part being manufactured is of sufficient quality. Sydney has substantial experience working with the parts and the test equipment. Sydney writes test procedures that outline the steps to execute the tests such that executing the tests should take 1 h on each part. Following these procedures, Sydney can verify the quality of each part in 1 h. Months later, Sydney is promoted and Jody is given the responsibility of testing the quality of the parts. Jody is new to the company and to the specific product line. Jody follows Sydney's test procedures, but Jody takes 2 h to test each part, instead of 1 h, reducing the overall throughput of the test team. Why?
>
> The test procedures were written at a level relevant to Sydney's use and not for someone with less experience in the product line. Doing so saved Sydney time but also increased the amount of time that someone unfamiliar with the testing would need to test each part, which introduced TD into the system. While Sydney saved time in creating the procedures, the system took on debt in the form of a less than ideal set of test procedures. The debt impacts the system when Jody takes longer to test each part and slows down the process. In this case, paying back the debt requires rewriting the test procedures such that they are at a sufficient level for any engineer, regardless of experience, to be able to use efficiently. The system suffered from delays due to the increased time to evaluate each part and also from the time to rewrite the test procedures. This TD can impact the project schedule, the cost of the project, and also the quality of the outputs. Was this example helpful in explaining TD?

This example highlights the major concept of TD: A technical compromise made to achieve a short-term benefit creates additional costs in the long term. To visually explain the concept of TD, Izurieta et al. [29] developed a conceptual map of TD for software engineering, which was extended by Rios et al. [13]. The concept maps visualize the major components of TD and associate these components with the system and business goals. While a useful aid in understanding the fundamental concepts of TD, these maps contain some notable deficiencies, including the lack of a feedback loop between TD and the system performance.

To address these concerns, a modified concept map of TD within the context of systems engineering has been developed based on a synthesis of TD components identified in the literature and interactions between the system and its stakeholders. This concept map is shown in Fig. 1. In this concept map, the business goals exert pressure on the system and its developers, who are then forced to make a technical compromise. This

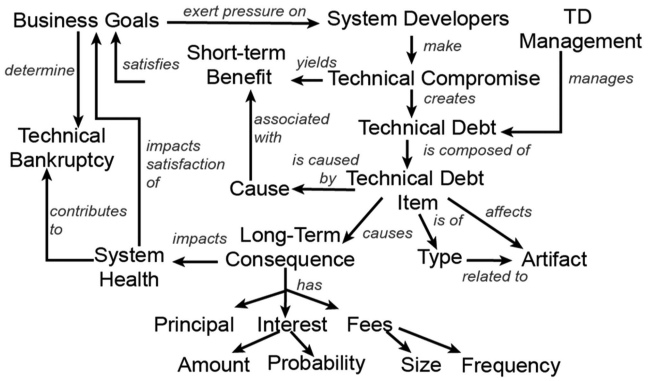

FIGURE 1. Conceptual map of TD for systems engineering, based on [13] and [29].

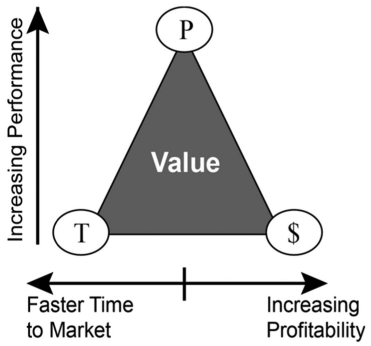

FIGURE 2. Interconnected system dimensions showing an estimation of system value.

technical compromise can yield a short-term benefit, which satisfies the business goals, but may create TD.

TD is composed of one or more technical debt items (TD Item), which have several attributes, including the affected artifact, the type of TD, the cause of the TD Item, and the long-term consequences. The cause is associated with the short-term benefit that satisfies the business goals. The long-term consequence is measured in principal, interest, and fees and impacts the system's health—the ability of the system to meet its performance objectives. These impacts on the system's health can also impact the satisfaction of the business goals, which leads to additional pressure on the system or to technical bankruptcy. TD management is an activity associated with the control of TD items. This map shows the feedback between TD and system health as an indicator of system performance.

The concept map defined in Fig. 1 provides a starting point for the creation of a TD ontology by identifying the relationship between the critical components of TD. This ontology is designed to provide common terminology and definitions that are focused on the systems engineering field. It leverages terminology from the software engineering field where possible. However, the ontology also redefines terms and introduces new terms as necessary to clarify the definitions and usages within systems engineering-specific applications.

B. BACKGROUND TERMINOLOGY

This section defines the background concepts and terminology used to establish the ontology.

1) SYSTEM DIMENSIONS

The development of a system can be characterized along three major dimensions: Budget, schedule, and performance, where performance is defined as the combination of the system's scope (what the system does) and its quality (how well it does it). These dimensions are linked together through a concept similar to the "Iron Triangle" of program management [30]: Stakeholders must conduct tradeoffs between the three dimensions. For example, the customer can define the scope through a requirement's specification and can define a delivery timeline. The system developer then determines the cost of the project which provides the developer with a sufficient value. The value is not necessarily a profit-driven parameter; a project may have other values to a system developer, such as the development of new technology. Alternatively, if the stakeholder asks for a product within a specified budget and on a specified schedule, then the scope of the deliverable may need to change. In this case, the stakeholder must conduct a tradeoff between the achievable scope and the available budget and schedule.

The triangle concept can be represented visually, as shown in Fig. 2, where the vertices are performance (P), profitability ($), and speed to market (T). The area of the triangle represents the value of the system. As the values of the dimensions change, the vertices will move, altering the system value. The farther the vertices are from the center of the triangle, the larger the area of the triangle and, therefore, the larger the value provided by the system: Faster time to market, higher profitability, and better performance, all deliver a higher value. Increased costs cause the profitability vertex to move left, which lowers the area of the triangle and decreases the overall value. Similarly, realized cost savings increase the profitability, moving the vertex to the right and increasing the overall value.

2) PHASES

System lifecycles flow through characteristic stages, regardless of the development strategy that is employed [31]. Different strategies result in different frequencies and numbers of iterations through the system lifecycle. Broadly speaking, the systems lifecycle can be divided into two phases: System development and system deployment. The development phase might consist of the following stages, adapted from [31]:

1) *Needs Analysis:* Definition of system capabilities to satisfy stakeholder needs;
2) *Requirements Definition:* Decomposition of stakeholder requirements into system requirements;
3) *Preliminary Design:* Development of design specifications to prove the ability of the system to meet requirements;
4) *Critical Design:* Detailed design of the system;

5) *Integration:* Implementation and integration of the components of the system;
6) *Verification:* Verification that the components and the integrated system meet the requirements.

The deployment phase might consist of the following stages:

1) *Validation:* Validation that the integrated system meets the stakeholders needs;
2) *Operations:* Postdevelopment phases of the system, consisting of production, use, maintenance, and retirement of the system.

Within this ontology, the development phase will be used to refer to the activities leading up to system validation and the deployment phase will be used to refer to activities that occur during and after system validation, including production, operations, maintenance, and retirement.

C. TECHNICAL DEBT DEFINITION

Cunningham [1] introduced the concept of TD stating the following.

"Although immature code may work fine and be completely acceptable to the customer, excess quantities will make a program unmasterable, leading to the extreme specialization of programmers and finally an inflexible product. Shipping a first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."

He used the term as a metaphor and not a definition. However, its introduction of the metaphor has proven useful in explaining the impact of technical decisions in terminology familiar to personnel who are not involved in the system development.

With its increased use, there is a need to provide a consensus definition of the TD metaphor. Many authors have provided definitions of TD, especially within the realm of software engineering. These definitions have subtle differences and nuances; however, the following components are common across the definitions.

1) TD occurs due to decisions made for short-term benefits that have long-term negative consequences [32], [33], [34], [35], [36], [37].
2) Taking on TD involves making a compromise in one area to achieve a benefit in another area (e.g., reducing the quality of testing to save schedule) [35], [38], [39], [40], [41], [42]
3) The effect of taking on TD is an increased amount of work in the future [38], [43], [44].

Some authors propose alternate definitions of TD, including presenting it as a gap between the actual and should-be state of a system [6], [12], [45], the existence of incomplete or immature components [7], [46], or work not yet done [2]. We assert that these definitions do not reflect the central tenant of Cunningham's initial concept that the decisions made today may result in increasing consequences tomorrow.

Rosser and Ouzzif [47] provided a systems engineering-based definition: "Expedient engineering decisions in requirements, architecture, design, documentation, integration, and test are made to gain short-term advantage, with similar negative effects on productivity and quality as have been shown in software." This definition is cumbersome and does not detail the negative effects, instead relying on foreknowledge of the application of TD within software engineering.

Jones et al. [48] defined TD as consisting of "design or implementation constructs that are expedient in the short term but that set up a technical context that can make a future change more costly or impossible." This definition does not define what an "expedient construct" is and whether it is due to poor design or intentional choices. Additionally, this definition states that TD only impacts the system when future changes are required. However, as will be discussed later, there is a component of TD associated with the use of a system.

To enable clear communication, a concise and easily understood definition of TD is preferred. Therefore, the definition of TD for systems engineering proposed by Kleinwaks et al. [22] is adopted here.

Definition 1: TD is a metaphor reflecting technical compromises that can yield short-term benefits but may hurt the long-term health of a system.

Definition 1 identifies the TD metaphor—the application of the concept of TD to describe potential problems within a system. The metaphor is used to talk about the abstract concept without necessarily relating it to concrete numbers and measurements. However, the term "technical debt" is also commonly used to refer to "the complete set of TD items" [49] within the system and as a value representing something the system "owes." When used in this context, the term TD takes on a different meaning, as listed in Definition 2. To limit the confusion, the term "TD metaphor" is used in the conceptual context and the term "technical debt" or "TD" is used in the quantitative context.

Definition 2: TD is the quantitative impact on the long-term health of the system accrued as the result of a technical compromise made to achieve a short-term benefit.

These definitions of TD consist of four main components: technical compromises, short-term benefits, the potential for negative impacts, and the long-term health of the system. The following sections explain these components in more detail.

1) TECHNICAL COMPROMISES

Referencing Fig. 2, compromises can be made that affect one or more of the system dimensions. For example, the stakeholder can compromise on budget by adding funding or compromise on schedule by delaying delivery until the system reaches the specified level of quality. Technical compromises are defined in Definition 3.

Definition 3: A technical compromise is a concession made in the performance dimension, either in scope or quality.

Only decisions that require concessions in the performance dimension are included in this definition. Decisions such as increasing the development timeline to enable the full realization of the system design (concession on schedule, benefit on performance) do not constitute TD.

2) SHORT-TERM BENEFIT

A system-level benefit is an increase in system capability in one of the three dimensions. A benefit in schedule would be the reduction in the time required to release a product. A benefit in performance would be the increase of capability in one area of the system. A short-term benefit is one that quickly realizes the benefit for the stakeholders and the system developers. For example, releasing a product two days earlier is a short-term benefit. A long-term benefit would be one that is not manifested until later in the system lifecycle. For example, an increase in system documentation may produce a benefit by reducing the complexity of system-level maintenance and a corresponding increase in the performance dimension. Generating the documentation during the system design phase results in a long-term realization of the benefit. The actual calendar times associated with short-term and long-term are subjective and dependent upon the system being developed.

Decisions that do not yield short-term benefits do not constitute TD. For example, the decision to invest in the development of a new factory instead of running additional shifts at the current factory provides a long-term benefit instead of a short-term benefit and, therefore, does not constitute TD.

3) POTENTIAL FOR NEGATIVE IMPACTS

Unlike financial debt, TD has intrinsic uncertainty about when it will need to be repaid and exactly how large the cost will be to repay the debt. Cunningham [1] captured this concept when he said "The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt." If system developers have to interact with the portion of the system that has TD, then they will have to expend additional effort to develop that portion of the system. However, if developers never interact with that component, then the technical compromise will not impact the system development. Definition 1 captures the probabilistic nature of TD—there is a chance that the debt will need to be repaid and also a chance that the debt may not negatively impact the system. The probabilistic nature of TD must be considered when making the initial technical compromise.

4) LONG-TERM HEALTH OF THE SYSTEM

The result of the technical compromise is often a long-term impact on the system's health if the technical concessions are not restored. Cunningham [1] recognized this fact when he stated "Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise." If the technical concessions that are made are left uncorrected, then the system's health may become compromised over time. Kothamasu et al. [50] defined the health of a deployed system as the ability of the system to stay in an operable condition, characterized by margins in design specifications, lack of observable damage to the system, system reliability, and performance parameters that are within the required bounds, and lack of any issues that would compromise the integrity of the system [51]. However, technical compromises affect both the development and operational phases of the system lifecycle [23], and therefore, an updated system of health definition covering both phases is required.

Definition 4: The system's health is the ability of the system to meet its objectives in the performance dimension without changes to the budget or schedule dimensions.

During system development, objectives in the performance dimension include designing and implementing the system in line with the scope and quality requirements. After the system is deployed, these objectives include meeting the quality requirements, such as usability and maintainability. A system that fails to meet either set of objectives within its planned schedule and budget is unhealthy. In development, an unhealthy system requires additional funds and/or schedules to deliver the required performance. After deployment, an unhealthy system underperforms its requirements, especially in areas of maintenance and usability.

To be considered TD, the impacts on the system's health must be long term. The use of the long-term qualifier implies that the impacts will remain in the system unless they are corrected. A short-term impact of a decision is resolvable and, if resolved, may have no significant impact on future changes. As such, this type of decision is an alternative design choice and does not incur TD [38]. For example, a test is specified to be conducted with a flight model of a satellite component. However, the component is delayed, and in order to keep the test schedule, an engineering model of the satellite component is used instead. This decision represents a technical compromise—the exact flight unit is not tested. However, if the engineering model is of sufficient quality, the test results will be valid and do not require retesting and there is no long-term impact.

5) IMPACT OF TECHNICAL DEBT ON THE SYSTEM DIMENSIONS

Fig. 3 shows the progressive impact of TD on the overall value of the system. Section 1 of the diagram shows the baseline system, with target performance (P), profitability ($), and speed to market (T) objectives resulting in a defined system value. TD affects the long-term health of the system through a concession made in the performance dimension. The system may still meet the overall scope requirements but the concessions may make additional changes more complicated. This system state is represented in section 2 of the diagram. Here, a small amount of TD has been introduced into the system (the

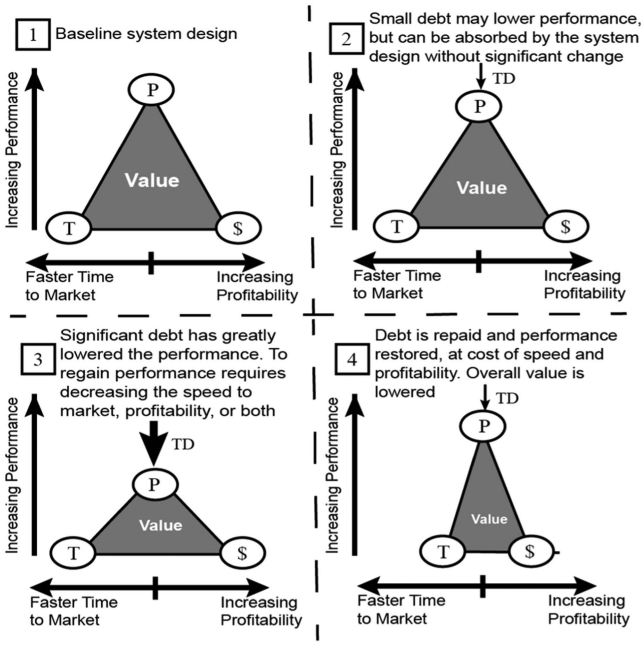

FIGURE 3. Impact of TD on project schedule, performance, and cost during project execution. Restoring system performance requires reducing the time to market or the profitability of the project.

arrow labeled TD) which does not have a significant impact on the overall value of the system. Section 2 represents system states such as taking on prudent deliberate debt, which is debt incurred with a known repayment plan [52], to meet a specified release date, where the reduction in performance is acceptable. In fact, not all TD taken on during the course of system development is detrimental as debt incurred intentionally to meet a deadline may benefit the system [53]. Prudent TD can be recovered in future releases.

In section 3 of Fig. 3, the TD has grown significantly, indicated by the larger arrow, drastically reducing both the system performance and the overall system value. The performance can no longer be recovered without adjustments to the other vertices. This state arises from the accumulation of TD either through reckless and inadvertent means, where TD is incurred without a defined repayment plan [52], or through the failure to follow the TD repayment plan. Section 4 of Fig. 3 shows how the system performance can be recovered by paying down the TD. The schedule and profitability vertices have both been moved inward to prop up the performance vertex, representing an increase in time to market (schedule delays) and lower profitability (increasing cost). While the performance has been restored, the overall value of the system (the area of the triangle) is reduced.

This analysis shows how TD, which is incurred in the performance dimension, can have impacts on the other system dimensions. The system dimensions are interconnected and, therefore, require a system-level view of TD in order to mitigate the impact of the debt. This ontology provides the communication framework necessary to support the system-level view.

D. SYSTEM TECHNICAL DEBT

"System TD" refers to the set of TD items present in the system. It is used to address the accumulation of TD from multiple sources within a system and separates the definition of the total TD (system TD) from the TD associated with each TD Item (technical debt). The system TD provides a method to quantitatively understand the accumulation of TD within the system.

Definition 5: System TD is the concrete set of TD items present in the system.

E. TECHNICAL DEBT MEASUREMENT UNITS

Addressing TD quantitatively requires that TD be measured in a consistent unit across all occurrences. A consistent measurement enables comparison of the impact of multiple TD items. TD has been measured as the financial cost [37], the amount of time required to do the work [54], and the work required to be performed [46]. Each of these terms has varying degrees of usability in systems engineering and roughly aligns with the three system dimensions of budget (financial cost), schedule (time required), and performance (work required). Definitions 1 and 3 state that TD starts with a technical compromise in the performance dimension of the system that results in impacts on the long-term health of the system. Definition 4 states that the health of the system is the ability to meet requirements in the performance dimension. The health of the system can be measured in the performance dimension by assessing the change in performance required to achieve the objectives. Since both the technical compromise and the long-term health can be measured in the performance dimension, TD should also be measured in the performance dimension.

The units used to measure the performance dimension will be different across different systems. For example, some systems may choose to measure the performance dimension based on the number of labor hours required to complete the work. Other systems may measure the performance dimension in terms of the lines of code that need to be written, and other systems may use the number of verified requirements. Still, other systems may convert the performance dimension into strict financial terms. With respect to TD, the specific unit does not matter as long as each TD Item is represented in the same unit for comparison and the unit used is understood by the system development team. Given the freedom of a system to report instances of TD in their own units, the term UNIT will be used within this ontology as the measurement of the TD.

Definition 6: The UNIT of TD is a quantified measurement of a change in the performance dimension of the system.

F. TECHNICAL BANKRUPTCY

Technical concessions may increase the cost of developing new features and the costs of maintaining the system [55]. The project development schedule may be exceeded due to the impact of the technical concessions. The impacts in the performance dimension may be so severe that the system

development cannot continue or that the maintainability and reliability of the deployed system are insufficient [40]. Reaching any of these conditions puts the system into a state of technical bankruptcy, defined as follows.

Definition 7: Technical bankruptcy is the state where the system can no longer proceed with its lifecycle until some, or all, of the system TD is repaid.

Systems that are technically bankrupt are no longer able to support future development without first repaying some or all of the existing system TD [5]. This situation can occur when the effort required to repay system TD exceeds the capacity of the development team. The development team will not be able to make progress on the system, resulting in a bankrupt state. Bankrupt systems are no longer able to either verify or validate their requirements within the system development timeline and budget [55].

A system may also reach technical bankruptcy once it is deployed due to an accumulation of technical fees. Technical fees, which define the increased difficulty in using the system due to the presence of unpaid principal, impact the quality of the delivered product. An excessive accumulation of fees will make the system unusable until the TD that resulted in the fees is corrected.

Systems reach technical bankruptcy when the technical costs, associated with system development and system, use exceed system benefits, such as delivering on time and within budget. Technically, bankrupt systems require an increase in the budget or schedule dimension or a reduction in the expectations in the performance dimension to emerge from bankruptcy.

G. TECHNICAL DEBT ITEM

A TD Item is a concrete instance of TD within a system that connects the technical concession and its consequences on system artifacts [49]. The TD item represents the concession that was made as part of the technical compromise and is used to track the impacts on the long-term health of the system. The following sections discuss each of the TD Item attributes.

1) DESCRIPTION

The description provides a narrative of the technical concession and the steps required to restore the system in the performance dimension.

2) CONSEQUENCE

The consequence of the TD item refers to the potential impacts on the long-term health of the system [29]. It consists of a narrative description of the impacts and the quantitative measures of principal, interest, and fees.

3) PRINCIPAL

The existing definitions of the principal vary but are typically centered on the effort required to correct the issue causing the TD [5], [54], [56], [57]. Ampatzoglou et al. [40] defined it as "the effort that is required to address the difference between the current and the optimal level of design-time quality." Izurieta et al. [58] stated that principal "refers to the cost or effort (measured monetarily or in time units) necessary to restore a software artifact back to health." Avgeriou et al. [49] identified principal as the "cost savings gained by taking some initial approach or "shortcut" in development (the initial principal). Or the cost it would take now to develop a different or better solution (the current principal)."

This quick review of the literature identifies two major methodologies for calculating the principal: It is either the UNIT to implement the optimal solution originally (the savings from the original concession) or it is the current UNIT to implement the optimal solution now. The principal measures the initial concession made as part of the technical compromise—it is the UNIT of system performance that is given up in order to achieve the desired benefit. The principal is like the principal in financial debt—it does not increase with time, although payments can be made to reduce the principal, and therefore, the following definition is adopted.

Definition 8: The principal P is a measurement of the concession made in the performance dimension to achieve a short-term benefit.

4) INTEREST AND FEES

The long-term impact of TD on a system's users is different than the impact on the system's developers. System users may experience decreased usability, maintainability, or reliability of the system. These impacts are likely to occur each time the system is used and to be of the same magnitude with each occurrence. System developers may experience the same issues but may also experience increased difficulties in continuing the system development to meet performance requirements. The impacts seen by developers occur when the system is modified either due to the natural development process or due to changes in the system requirements. These impacts tend to be less predictable both in occurrence and in the magnitude of the impact. Therefore, the long-term impact of the technical concession needs to be considered from both perspectives. This consideration results in two separate quantities: interest and fees.

Interest and fees can be distinguished based on how they impact the system. Interest is based on impacts on the development of the system—if the technical concession results in increased costs, schedule, or difficulty in making modifications to the system, then the system has accrued interest. Interest is variable—both the interest amount and the interest probability are functions of the state of the system. Fees are based on impacts observed during usage of the system—if the system is more difficult or complicated to use as a result of the technical concession, then the system has incurred a fee. The magnitude of the fee is constant; however, the fee must be paid by the user every time that the impacted artifact is used.

Fees are paid by the user and the interest is paid by the system developer. The system developer must repay the principal. The repayment of the principal constitutes the correction of

the original technical concessions and removes the TD. This repayment must be done by the developers and then the updated system is released to the users.

a) Interest: Interest on TD is traditionally defined as the extra effort required to modify the system due to the presence of deficiencies [5], [32], [36], [40], [54]. TD interest has also been defined as the work to correct a deficiency [58], the additional work to implement new functionality [49], and the additional work to maintain the system due to the presence of the deficiency [55], [59].

TD interest results from the lower design-time quality of a component (poor documentation, low maintainability, etc.) that requires additional effort in subsequent development efforts. TD interest can be contagious [20]—each new component that interacts with a component containing TD may require additional effort to develop and may then carry forward that interest into its successor components (compounding the interest). If a suboptimal component is included in the architecture instead of correcting the component (the principal), each new application that connects to that component would suffer from its suboptimality [56]. The build-up of dependencies on the suboptimal component results in overall suboptimal performance and increased work to add new components (the interest).

Within the systems engineering context, the TD interest refers to the long-term impacts on the system as encountered by the system developers. The interest will accrue in the performance dimension of the system and can impact both the scope and the quality of the system. The interest definition, therefore, is limited to the impact on the system developers.

Definition 9: The interest I is the expected value of additional UNITs incurred by the system developers in the performance dimension due to the presence of unpaid principal.

Applying a direct financial analog to TD would require the definition of an "interest rate" for TD, which would associate the total amount to be repaid with a known growth rate of the debt. However, a relationship between TD principal and interest that would apply to all projects has not been defined. Ampatzogolou et al. [54] suggested that such a rate cannot be defined, since the specific growth of TD interest depends on aspects unique to each system, such as the system implementation, the system context, and the maintenance activities performed. Due to the complexities in calculating and predicting the effort associated with interest, Seaman et al. [39] divided the interest into two categories: interest amount and interest probability.

b) Interest amount: The interest amount reflects the long-term change in the performance dimension that is traceable to the original concession (the principal) [39].

Definition 10: The interest amount a is the additional UNITs incurred by the system developers in the performance dimension due to the presence of unpaid principal as a function of the state of the system.

The interest amount represents the impact of the principal on the future state of the system. For example, if the principal was incurred due to a decision to not complete documentation, then the interest amount would be an increase in the effort required to update that part of the system in future iterations. The interest amount is measured in the same UNIT as the principal. The interest amount is a function of the state of the system development and the development timeline. For example, a system may initially have few interfaces and components, and the ability to work around the initial concession is small. As the system grows, the number of interfaces increases and the impact of the initial concession spreads to a larger number of interfaces and components. Therefore, the change in the performance dimension has increased due to the larger number of impacted components, increasing the interest amount. The interest amount may also decrease due to changes in the system development, such as removing an interface.

c) Interest probability: Unlike financial debt, which has a known schedule of payments and interest, TD interest may or may not be realized. Once the technical compromise is made, the principal exists in the system. The technical compromise may be made in a component of the system that never has to be altered again, and therefore, the compromise does not need to be resolved. The interest probability accounts for the likelihood of the interest being realized [39]. For example, a system may choose smaller batteries that reduce the upgradability of the system. However, if those upgrades are not implemented, then the interest is never realized.

Definition 11: The interest probability r is the probability that the interest amount will be realized as a function of the state of the system.

Like the interest amount, the interest probability is also a function of the state of the system and the development timeline. As the system development changes, especially in iterative design cycles, the probability of the interest being realized may change. For example, an incompletely implemented standard may initially have a low-interest probability if the component that implements the standard is isolated. If the system design changes such that the component is no longer isolated, then the interest probability will increase.

d) Fees: Users of a system use a released version of the system, and therefore, the system's capability in the performance dimension is largely fixed. Design choices made in the system development may result in a less-than-optimal experience for the user. Activities may take longer than they should due to underperforming hardware or due to poor user interface design. Capabilities may not be fully implemented and require workarounds by the user. The system may not be easily maintained or may not be as reliable as it should have been. All these issues tend to occur in the quality aspect of the performance dimension. Unlike TD interest, these issues occur with each use of the system. The total impact of the issues is dependent upon the number of times that the system is used and is not based on the effort required to add capability to the system or to modify the system design.

Therefore, these impacts on the health of the system are separated out from the TD interest and are instead termed TD

fees. Fees are the recurring costs of using a system containing TD and are measured in UNIT every time that the system is used. Izurieta et al. [29] defined this concept as *recurring interest*. An example of a fee occurs when a poorly developed user interface results in several extra minutes spent inputting system parameters in a software system. Every time the user has to input the parameters, users will have to "pay the fee" for those extra minutes, until the principal on that TD Item (reworking the interface) is repaid by the developers. A fee is defined as follows.

Definition 12: The fee f is the amount of additional UNIT incurred by the user with each use of the system due to the presence of TD.

A system that performs poorly does not necessarily have fees. Fees must be associated with a technical compromise, and as such, an instance of TD. For example, a race car that is slower than its competitors does not necessarily have any fees associated with the use of the car—it is just not as well designed as its competitors. However, a cost savings compromise made to use a metal frame instead of a composite frame that reduces the gas mileage would be an example of a fee—the user (the driver) must perform additional pit stops every time the car is raced.

5) BALANCE

The balance B is the summation of the principal and the interest and represents the total UNITs required to repay the TD item. The balance does not include the fees (either realized or anticipated) in the system, as fees are not repaid. The expected value of the balance is calculated, as given in (1). The subscript t indicates the parameters that change with time.

$$\overline{B_t} = P + a_t * r_t. \tag{1}$$

6) TOTAL COST

The total cost C in terms of UNIT, due to the TD item, is inclusive of the balance and the fees. The total cost is a time-dependent value, as it includes the interest, which is a function of the state of the system, and the expected fees. Fees are fixed in magnitude, but the number of fees n will change with time. The expected value of the total cost is calculated, as given in (2). The subscript t indicates the parameters that change with time.

$$\overline{C_t} = B_t + f * n_t. \tag{2}$$

7) ARTIFACT

The artifact is the part of the system that is affected by the TD [29]. A TD item may impact multiple artifacts—the principal may be associated with one artifact while the interest and fees may be associated with a different artifact. An artifact may be a piece of documentation, a component of the system, a test case, or any other part of the system itself.

Definition 13: An artifact is the part of the system affected by TD.

8) CAUSE

The cause of a TD item defines the reasons why the technical compromise was made [29]. It consists of two attributes: The specific cause and the cause category. The cause provides traceability of the TD item to the original decision that can then be used in forensic evaluations.

a) Specific cause: The specific cause of a TD item is the short-term benefit provided to the system developers, stakeholders, or users that is realized through a technical concession. The specific cause includes the rationale for why achieving the short-term benefit required a technical concession. For example, a technical compromise may be made such that a program increment can be released on time. In this example, the specific cause is the on-time release of the program increment. The rationale defines why a technical concession had to be made to release the increment on schedule, such as supply chain issues forcing a switch to a different less reliable part.

Definition 14: The specific cause of a TD item is the short-term benefit realized through the technical concession.

b) Cause category: The cause category provides a general categorization of the cause. The cause category is defined as follows.

Definition 15: A cause category is the dimension of the system development where the short-term benefits are achieved as a result of the technical concession.

Kleinwaks et al. [23] conducted an empirical survey of systems engineering professionals. This survey included questions on reasons why a system developer may incur TD. Over 80% of the respondents identified schedule pressure as a reason, over 60% of the respondents identified cost pressure as a reason, and over 30% of the respondents identified technical compromise as a reason. Kruchten et al. [17] similarly identified schedule pressure as the primary cause of TD. These results lead to the following cause categories.

1) *Schedule:* It consists of pressures put on the technical solution due to the need for the system to meet the schedule. For example, any TD incurred such that the system can meet its scheduled release date is caused by the schedule category.
2) *Cost:* It consists of pressures put on the technical solution due to the need for the system to stay on budget. For example, technical concessions associated with the use of a cheaper part are associated with the cost category.
3) *Performance:* It consists of technical concessions made in one area to achieve technical benefits in another area of the system. For example, a satellite system may choose to use a less performant antenna such that system mass requirements are met.

These categories mirror the system dimensions defined in Section II-B. As evidenced by Fig. 3, pressure on any of the dimensions may result in movement in the other dimensions. Fig. 4 shows an example of this process. Stakeholders, such as management executives, may put pressure on the system to release earlier in order to beat a competitor to market. This

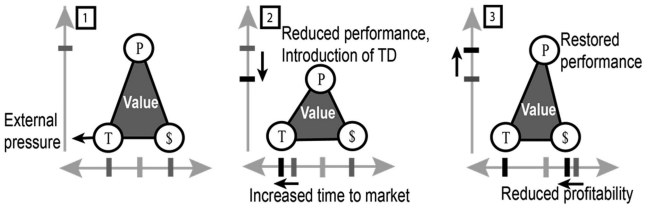

FIGURE 4. Example of schedule pressure creating TD.

pressure pulls the schedule vertex (T) to the left, as shown in section 1 in Fig. 4. Without other resources, the movement of the schedule vertex would result in a corresponding decrease in system performance (P), as shown in section 2 of Fig. 4. This reduction indicates that a technical compromise is required, which introduces TD to the system. To restore the system performance, the cost vertex ($) is moved left, decreasing the system profitability, as shown in section 3 of Fig. 4. Therefore, the stakeholders would have to accept a tradeoff in the system—either a decreased performance and the introduction of TD or decreased profitability due to an increase in costs. Note that the profitability factor here does not account for the potential future benefits of releasing the system earlier to the market.

9) TYPE

The type of a TD item provides a means to categorize the TD item. TD items with similar types may have similar causes or similar methods for repaying the TD. Examples of different types of TD can be found throughout the literature and include items such as architectural TD [60], domain debt [42], and requirements debt [61]. TD occurs in various stages throughout a system's lifecycle and for various reasons. Classification of TD into different types assists in understanding and managing it; however, too many disparate types of risk are diluting the strength of the TD metaphor [17]. A definition for a TD type, such as that provided in Definition 16, can assist in restricting the accumulation of differing TD types.

Definition 16: A TD type is a classification of TD based on the artifacts that are negatively affected by the technical concessions made to realize a short-term benefit.

This definition restricts a TD type to be associated with specific artifacts and the technical concessions that are made. Domain debt, defined as the "misrepresentation of the application domain by an actual system" [42], is associated with the documentation of stakeholder needs and system requirements. Technical concessions that result in domain debt can include limiting user interactions to save development time. Defect debt, defined as any defect found within the system [5], would not be a type of TD according to Definition 16. Defect debt can be mapped to an artifact, such as the source code, but not to technical concessions. Defects are the result of poor work and are not inserted into the system to realize a short-term benefit.

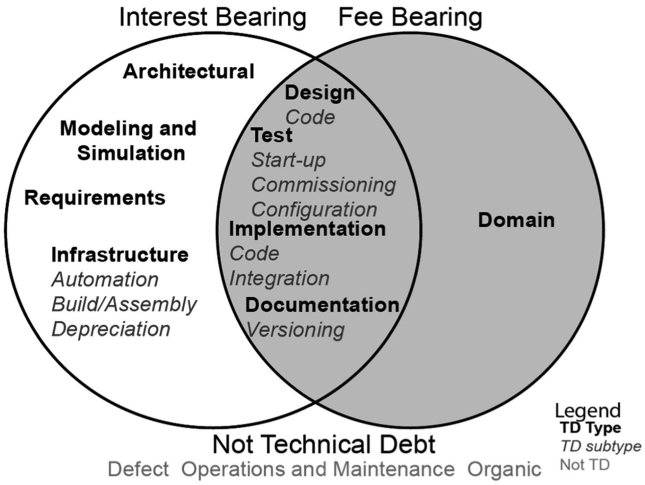

FIGURE 5. Consolidation of TD types from [22], organized by interest and fee-bearing status.

III. DISCUSSION

The ontology provided in this article provides a starting point for developing a common framework for discussing TD within systems engineering. This commonality is critical to enable the sharing of methods and processes for identifying and mitigating the impacts of TD. The need for a common set of definitions can be seen by examining a listing of types of TD.

Kleinwaks et al. [22] identified the types of TD found within published systems engineering research. Recognizing that creating too many types of TD risks can dilute the strength of the metaphor [17], the types of systems engineering TD were reevaluated in the context of this TD ontology. This evaluation resulted in the consolidation of the TD types, as shown in Fig. 5, with the types classified as interest bearing (associated primarily with impacts during system development), fee bearing (associated primarily with impacts during system usage), or both interest and fee bearing. Several of the identified types of TD proved to be instances of other types of TD. For example, versioning debt is an instance of documentation debt and not a separate type of TD. Automation debt, build/assembly debt, depreciation debt, and infrastructure debt were originally listed as different types of TD [22]. These types of TD impact the same artifacts—the supporting tools used to develop the system. Therefore, according to ontology, they represent different facets of the same type of TD. Fig. 5 shows the subtypes as italicized items under the new parent type, which is listed in bold. After application of the definition of a TD type, several types of TD listed in [22] were found to not be TD types: defect, operations and maintenance, and organic. These items reflect the causes or impacts of TD instead of types of TD. This short example demonstrates the utility of the ontology—it provides clear guidelines of what is and is not TD and can prevent overclassification, which impedes communication and the development of effective management strategies [17].

IV. CONCLUSION

Kleinwaks et al. [22] proposed a research agenda for understanding TD in the context of systems engineering. This agenda includes baselining the knowledge of TD in the field of systems engineering through empirical data collection, developing a systems engineering ontology of TD, developing techniques to identify causes of TD within systems engineering, developing methods to quantify and predict the impact of TD within systems development, and verifying and validating these methods.

The first agenda item was addressed through a survey on the prevalence of TD in systems engineering [23]. The research question presented in this article addresses the second agenda item—identifying an ontology of TD within the context of systems engineering. This research presents a starting point for the development of a complete ontology. It introduces and defines the key terms, with clear explanations. These explanations and definitions begin the creation of a common lexicon and provide practitioners with the semantics necessary to create clarity in communications.

The ontology presented here is not complete. Further work needs to be performed to create taxonomies of TD types and of specific causes of TD. Too many classes of either TD types or specific causes can dilute the strength of the TD metaphor [17]. Future research needs to provide guidance on how to classify TD such that it is precise enough to be meaningful without overspecification.

The socialization of this ontology, of which this article is the first step, will provide a starting point for clear and concise terminology usage within the field of systems engineering, which is a necessary step toward mitigation of the risks associated with TD and the prevention of technical bankruptcy.

IV. REFERENCES

[1] W. Cunningham, "The WyCash portfolio management system," Mar. 26, 1992, Accessed: Jan. 29, 2022. [Online]. Available: http://c2.com/doc/oopsla92.html

[2] L. Wheatcraft, T. Katz, M. Ryan, and R. B. Wolfgang, "Needs, requirements, verification, validation lifecycle manual," *INCOSE Requirements Working Group*, Tech. Prod. INCOSE-TP-2021.002-01, Jan. 2022.

[3] Project Management Institute, "Technical debt," Feb. 2022, Accessed: Mar. 11, 2023. [Online]. Available: https://www.pmi.org/disciplined-agile/agile/technicaldebt

[4] V. Dalal, K. Krishnakanthan, B. Munstermann, and R. Patenge, "Tech debt: Reclaiming tech equity," Oct. 2020. Accessed: Mar. 12, 2023. [Online]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/tech-debt-reclaiming-tech-equity

[5] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *J. Syst. Softw.*, vol. 101, pp. 193–220, 2015.

[6] R. E. Fairley and M. J. Willshire, "Better now than later: Managing technical debt in systems development," *Computer*, vol. 50, no. 5, pp. 80–87, 2017.

[7] R. S. Carson, P. J. Frenz, and E. O'Donnell, "Project manager's guide to systems engineering measurement for project success: A basic introduction to systems engineering measures for use by project managers (Version 1.0)," INCOSE, San Diego, CA, USA, 2015.

[8] D. A. Broniatowski and J. Moses, "Measuring flexibility, descriptive complexity, and rework potential in generic system architectures," *Syst. Eng.*, vol. 19, no. 3, pp. 207–221, 2016.

[9] A. T. Bahill, "Diogenes, a process for identifying unintended consequences," *Syst. Eng.*, vol. 15, no. 3, pp. 287–306, 2012.

[10] S. Dullen, D. Verma, and M. Blackburn, "Review of research into the nature of engineering and development rework: Need for a systems engineering framework for enabling rapid prototyping and rapid fielding," *Procedia Comput. Sci.*, vol. 153, pp. 118–125, 2019.

[11] L. A. Rosser and J. H. Norton, "A systems perspective on technical debt," in *Proc. IEEE Aerosp. Conf. (50100)*, 2021, pp. 1–10.

[12] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *J. Syst. Softw.*, vol. 86, pp. 1498–1516, 2013.

[13] N. Rios, M. G. de Mendonca Neto, and R. O. Spinola, "A tertiary study on technical debt: Types, management strategies, research," *Inf. Softw. Technol.*, vol. 102, pp. 117–145, 2018.

[14] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spinola, "Towards an ontology of terms on technical debt," in *Proc. IEEE 6th Int. Workshop Manag. Tech. Debt*, 2014, pp. 1–7.

[15] M. Uschold and R. Jasper, "A framework for understanding and classifying ontology applications," in *Proc. Workshop Ontol. Problem-Solving Methods*, 1999, pp. 11-1–11-12.

[16] Y. Yang, D. Verma, and P. S. Anton, "Technical debt in the engineering of complex systems," *Syst. Eng.*, vol. 26, pp. 590–603, 2023.

[17] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Softw.*, vol. 29, no. 6, pp. 18–21, Nov./Dec. 2012.

[18] N. S. Alves, T. S. Mendes, M. G. de Mendonca, R. O. Spinola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," *Inf. Softw. Technol.*, vol. 70, pp. 100–121, 2016.

[19] D. Pina, A. Goldman, and G. Tonin, "Technical debt prioritization: Taxonomy, methods results, and practical characteristics," in *Proc. IEEE 47th Euromicro Conf. Softw. Eng. Adv. Appl.*, 2021, pp. 206–213.

[20] A. Martini and J. Bosch, "The danger of architectural technical debt: Contagious debt and vicious circles," in *Proc. IEEE/IFIP 12th Work. Conf. Softw. Architecture*, 2015, pp. 1–10.

[21] S. Malakuti and J. Heuschkel, "The need for holistic technical debt management across the value stream: Lessons learnt and open challenges," in *Proc. IEEE/ACM Int. Conf. Tech. Debt*, 2021, pp. 109–113.

[22] H. Kleinwaks, A. Batchelor, and T. H. Bradley, "Technical debt in Systems Engineering - A systematic literature review," *Syst. Eng.*, vol. 26, pp. 675–687, 2023.

[23] H. Kleinwaks, A. Batchelor, and T. H. Bradley, "An empirical survey on the prevalence of technical debt in systems engineering," in *Proc. INCOSE Int. Symp.*, 2023, pp. 1640–1658.

[24] S. Koolmanojwong and J. A. Lane, "Enablers and inhibitors of expediting systems engineering," *Procedia Comput. Sci.*, vol. 16, pp. 483–491, 2013.

[25] G. Robiolo, E. Scott, S. Matalonga, and M. Felderer, "Technical debt and waste in non-functional requirements documentation: An exploratory study," in *Proc. Int. Conf. Product-Focused Softw. Process Improvement*, 2019, pp. 220–235.

[26] J. A. Lane, S. Koolmanojwong, and B. Boehm, "4.6.3 Affordable systems: Balancing the capability, schedule, flexibility, and technical debt tradespace," in *Proc. INCOSE Int. Symp.*, 2013, pp. 1385–1399.

[27] B. W. Boehm, J. A. Lane, S. Koolmanojwong, and R. Turner, *The Incremental Commitment Spiral Model: Principles and Practices for Successful Systems and Software*, Upper Saddle River, NJ, USA: Addison-Wesley, 2014.

[28] S. Blumberg, R. Das, J. Lansing, N. Motsch, B. Munstermann, and R. Patenge, "Demystifying digital dark matter: A new standard to tame technical debt," Jun. 2022. Accessed: Mar. 12, 2023. [Online]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/demystifying-digital-dark-matter-a-new-standard-to-tame-technical-debt

[29] C. Izurieta et al., "Perspectives on managing technical debt: A transition point and roadmap from Dagstuhl," in *Proc. Joint 4th Int. Workshop Quantitative Approaches Softw. Qual., 1st Int. Workshop Tech. Debt Analytics*, 2016, pp. 84–87.

[30] J. W. Boswell, F. T. Anbari, and J. W. Via, III, "Systems engineering and project management: Points of intersection, overlaps, and tensions," in *Proc. Portland Int. Conf. Manage. Eng. Technol.*, 2017, pp. 1–6.

[31] A. Kossiakoff, S. J. Seymour, D. A. Flanigan, and S. M. Biemer, *Systems Engineering: Principles and Practice*, 3rd ed. Hoboken, NJ, USA: Wiley, 2020.
[32] T. Besker, A. Martini, and J. Bosch, "Managing architectural technical debt: A unified model and systematic literature review," *J. Syst. Softw.*, vol. 135, pp. 1–16, 2017.
[33] R. Verdecchia, P. Kruchten, and P. Lago, "Architectural technical debt: A grounded theory," in *Proc. Eur. Conf. Softw. Architecture*, 2020, pp. 202–219.
[34] K. Borowa, A. Zalewski, and A. Saczko, "Living with technical debt – A perspective from the video game industry," *IEEE Softw.*, vol. 38, no. 6, pp. 65–70, Nov./Dec. 2021.
[35] F. Ocker, M. Seitz, M. Oligschlager, M. Zou, and B. Vogel-Heuser, "Increasing awareness for potential technical debt in the engineering of production systems," in *Proc. IEEE 17th Int. Conf. Ind. Inform.*, 2019, pp. 478–484.
[36] V. Lenarduzzi and D. Fucci, "Towards a holistic definition of requirements debt," in *Proc. IEEE/ACM Int. Symp. Empirical Softw. Eng. Meas.*, 2019, pp. 1–5.
[37] N. Brown et al., "Managing technical debt in software-reliant systems," in *Proc. FSE/SDP workshop Future Softw. Eng. Res.*, 2010, pp. 47–52.
[38] Z. S. H. Abad and G. Ruhe, "Using real options to manage technical debt in requirements engineering," in *Proc. IEEE 23rd Int. Requirements Eng. Conf.*, 2015, pp. 230–235.
[39] C. Seaman et al., "Using technical debt data in decision making: Potential decision approaches," in *Proc. IEEE 3rd Int. Workshop Manag. Tech. Debt*, 2012, pp. 45–48.
[40] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review," *Inf. Softw. Technol.*, vol. 64, pp. 52–73, 2015.
[41] S. McConnell, "Managing technical debt," *Construx Softw. Builders*, pp. 1–14, 2008.
[42] H. Storrle and M. Ciolkowski, "Stepping away from the lamppost: Domain-level technical debt," in *Proc. IEEE 45th Euromicro Conf. Softw. Eng. Adv. Appl.*, 2019, pp. 325–332.
[43] D. Sculley et al., "Machine Learning: The high-interest credit card of technical debt," in *Proc. SE4ML: Softw. Eng. Mach. Learn.*, 2014, pp. 1–9.
[44] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the size, cost, and types of technical debt," in *Proc. IEEE 3rd Int. Workshop Manag. Tech. Debt*, 2012, pp. 49–53.
[45] J. Schutz and M. Uslar, "Introducing the concept of technical debt to smart grids: A system engineering perspective," in *Proc. 25th Int. Conf. Electricity Distrib.*, 2019, pp. 1–5.
[46] C. Seaman and Y. Guo, "Measuring and monitoring technical debt," *Adv. Comput.*, vol. 82, pp. 25–46, 2011.
[47] L. A. Rosser and Z. Ouzzif, "Technical debt in hardware systems and elements," in *Proc. IEEE Aerosp. Conf. (50100)*, 2021, pp. 1–10.
[48] C. L. Jones, G. Draper, B. Golaz, and P. Januz, "Practical software and systems measurement continuous iterative development measurement framework. Part 3: Software assurance and technical debt version 2.1," *Nat. Defense Ind. Assoc., Int. Council Syst. Eng.*, 2021.
[49] P. Avgeriou, P. Kruchten, R. L. Nord, I. Ozkaya, and C. Seaman, "Reducing friction in software development," *IEEE Softw.*, vol. 33, no. 1, pp. 66–73, Jan./Feb. 2016.
[50] R. Kothamasu, S. H. Huang, and W. H. VerDuin, "System health monitoring and prognostics—A review of current paradigms and practices," *Int. J. Adv. Manuf. Technol.*, vol. 28, no. 9, pp. 1012–1024, 2006.
[51] D. H. Collins, C. M. Anderson-Cook, and A. V. Huzurbazar, "System health assessment," *Qual. Eng.*, vol. 23, no. 2, pp. 142–151, 2011.
[52] M. Fowler, "TechnicalDebtQuadrant," Oct. 14, 2009, Accessed: Jan. 27, 2022. [Online]. Available: https://martinfowler.com/bliki/TechnicalDebtQuadrant.html
[53] P. Kruchten, R. L. Nord, I. Ozkaya, and D. Falessi, "Technical debt: Towards a crisper definition report on the 4th international workshop on managing technical debt," *ACM SIGSOFT Softw. Eng. Notes*, vol. 38, no. 5, pp. 51–54, 2013.
[54] A. Ampatzoglou et al., "Exploring the relation between technical debt principal and interest: An empirical approach," *Inf. Softw. Technol.*, vol. 128, 2020, Art. no. 106391.
[55] M. Ciolkowski, V. Lenarduzzi, and A. Martini, "10 Years of technical debt research and practice: Past, present, and future," *IEEE Softw.*, vol. 38, no. 6, pp. 24–29, Nov./Dec. 2021.
[56] V. Lenarduzzi, T. Besker, D. Taibi, and A. Martini, "A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools," *J. Syst. Softw.*, vol. 171, 2021, Art. no. 110827.
[57] A. Martini, J. Bosch, and M. Chaudron, "Investigating architectural technical debt accumulation and refactoring over time: A multiple case study," *Inf. Softw. Technol.*, vol. 67, pp. 237–253, 2015.
[58] C. Izurieta, G. Rojas, and I. Griffith, "Preemptive management of model driven technical debt for improving software quality," in *Proc. 11th Int. ACM SIGSOFT Conf. Qual. Softw. Architectures*, 2015, pp. 31–36.
[59] G. Digkas, A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, and O. Matel, "The risk of generating technical debt interest: A case study," *SN Comput. Sci.*, vol. 2, no. 1, pp. 1–12, 2020.
[60] R. Verdecchia, I. Malavolta, and P. Lago, "Architectural technical debt identification: The research landscape," in *Proc. ACM/IEEE Int. Conf. Tech. Debt*, 2018, pp. 11–20.
[61] N. A. Ernst, "On the role of requirements in understanding and managing technical debt," in *Proc. 3rd Int. Workshop Manag. Tech. Debt*, 2012, pp. 61–64.

HOWARD KLEINWAKS received the B.Sc. and M.Sc. degrees in aerospace engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2003 and 2005, respectively. He is currently working toward the Doctor of Engineering degree in systems engineering with Colorado State University, Fort Collins, CO, USA.

He is researching the management of technical debt in iterative systems development. He is the Chief Engineer of the Strategic Space Business Unit at Modern Technology Solutions, Inc., and has more than 15 years of experience in the aerospace and systems engineering fields. He is a Project Management Professional, Associate Systems Engineer, and a Professional Scrum Master.

ANN BATCHELOR received the B.S. degree in chemistry from Erskine College, in 1973, and M.S. degree in nutrition from Clemson University, in 1975.

She has extensive industrial experience in technical management, systems engineering, production, manufacturing, lean engineering, life cycle management, test and analysis, transitioning technology into manufacturing, proposal and project management, and technical writing. Her experience includes more than 20 years in research, product development, manufacturing, engineering management, and business development in the defense industry as a Chief Scientist, Systems Engineer, Director of engineering, and Director of program management. While at CSU, she worked with the Energy Institute in natural gas leak monitoring projects and transportation projects. She teaches engineering project and program management, systems requirements engineering, and engineering risk assessment. She is a past certified Program Management Professional by the Project Management Institute, a Military Sensing Fellow (DOD Informational and Analysis Center for Military Sensing), a Lecturer on risk and opportunity management courses, a former president-elect of the International Council on Systems Engineering Atlanta Chapter, and a former course leader at GT for infrared and visible signature suppression course.

THOMAS H. BRADLEY (IEEE, Member) received the B.S. degree and another B.S. degree in mechanical engineering from the University of California—Davis, Davis, CA, USA, in 2000 and 2003, respectively, and the Ph.D. degree in mechanical engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2008.

He serves as the Woodward Foundation Professor and is the Department Head for the Department of Systems Engineering with Colorado State University, Fort Collins, CO, USA. He is a member of INCOSE, SAE, ASME, and AIAA. He conducts research and teaches a variety of courses in system engineering, multidisciplinary optimization, and design. His research interests are focused on applications in automotive and aerospace system design, energy system management, and lifecycle assessment.