Asset Management Taxonomy: A Roadmap

Ehsan Zabardast[a,*], Javier Gonzalez-Huerta[a], Tony Gorschek[a], Darja Šmite[a], Emil Alégroth[a] and Fabian Fagerholm[a,b]

[a]*Software Engineering Research Lab SERL, Blekinge Institute of Technology, Campus Gräsvik, Valhallavägen 1, Karlskrona, Sweden*
[b]*Department of Computer Science, Aalto University, Espoo, Finland*

ARTICLE INFO

Keywords:
Taxonomy
Asset Management in Software Engineering
Assets for Software-Intensive Products and Services
Technical Debt
Value

ABSTRACT

Developing a software-intensive product or service can be a significant undertaking, associated with unique challenges in each project stage, from inception to development, delivery, maintenance, and evolution. Each step results in artefacts that are crucial for the project outcome, such as source-code and supporting deliverables, e.g., documentation.

Artefacts which have inherent value for the organisation are assets, and as assets, they are subject to degradation. This degradation occurs over time, as artefacts age, and can be more immediate or slowly over a period of time, similar to the concept of technical debt. One challenge with the concept of assets is that it seems not to be well-understood and generally delimited to a few types of assets (often code-based), overlooking other equally important assets.

To bridge this gap, we have performed a study to formulate a structured taxonomy of assets. We use empirical data collected through industrial workshops and a literature review to ground the taxonomy. The taxonomy serves as foundations for concepts like asset degradation and asset management. The taxonomy can help contextualise, homogenise, extend the concept of technical debt, and serves as a conceptual framework for better identification, discussion, and utilisation of assets.

1. Introduction

The fast pace of software-intensive product and service development (SIPS) impacts the decision-making process both in design and operational decisions. Acting fast to cope with change will compromise the values of the delivered product, environment, development process, and the artefacts involved, such as source code, test cases, and documentation [14]. Though the decisions that are made under these circumstances focus on delivery, they might impact the quality of the delivered product or service, causing degradation of supporting artefacts such as code, architecture, and documentation, to name a few. Researchers and practitioners have traditionally used the Technical Debt (TD) [18] metaphor to refer to the impact of the intentional and unintentional sub-optimal decisions on assets as a consequence to meet deadlines [35]. However, the original TD metaphor focuses on code-based artefacts and often does not take non-technical assets such as documentation, test-cases, and supporting items into account [4]. This is evident from, for example, the tools available for measuring TD [55].

To elaborate and move towards a more holistic view of artefacts, more relevantly described as "assets" (having inherent value to the organisation in their efforts) [77], away from studying them in isolation [55], this paper presents the first version of a taxonomy, expandible as the field matures, of assets that are relevant for companies during the inception, planning, realisation, and evolution of SIPS. We define the term "asset" in the software engineering context by refining the definition in [ISO5500:2014] [30], as follows: **an asset is *"any artefact with an inherent value for the organisation, and that is used for, and/or enables the inception, development, delivery, or evolution of software-intensive products or services over time"*** [77].

From a practitioner perspective, the degradation of assets is central as its accumulation slows down the product's maintenance and evolution [4]. Degradation is compounded by the increase of SIPS complexity and size in general, even without the degradation [43, 42]. The need for further research into assets and asset degradation is also evident from the empirical study presented in this paper and the fact that assets are often not associated with the traditional technical debt concept despite being deemed critical to the deliverable product/service itself.

From a research perspective, the benefit of widening the term of an implicit "debt" to assets is that it takes the so-called "items of value" [49] into account and widens the view of what can hold value in a SIPS development context. This widening fosters an understanding of what artefacts can be negatively impacted by degradation, making it harder to develop and evolve SIPS. To this end, having a common vocabulary and structure of potential assets – a taxonomy of assets is presented – and the concepts of value and degradation are explored [77]. Besides, the purpose of a taxonomy is to identify knowledge gaps and interrelationships between objects and support decision-making [72, 71, 74, 70]. This

This research was supported by the KKS foundation through the SHADE KKS Hög project with ref: 20170176 and the KKS S.E.R.T. Research Profile with ref. 2018010 project at Blekinge Institute of Technology, SERLSweden.

*Corresponding author

ehsan.zabardast@bth.se (E. Zabardast); javier.gonzalez.huerta@bth.se (J. Gonzalez-Huerta); tony.gorschek@bth.se (T. Gorschek); darja.smite@bth.se (D. Šmite); emil.alegroth@bth.se (E. Alégroth); fabian.fagerholm@bth.se (F. Fagerholm)

https://ehsanzabardast.com/ (E. Zabardast); http://gonzalez-huerta.net/ (J. Gonzalez-Huerta); https://gorschek.com/ (T. Gorschek)

ORCID(s): 0000-0002-1729-5154 (E. Zabardast); 0000-0003-1350-7030 (J. Gonzalez-Huerta); 0000-0002-7298-3021 (F. Fagerholm)

work is mostly based on related work and industrial cases, i.e., companies addressing challenges related to asset degradation.

The taxonomy presented in this paper is based on the results acquired through a an empirical study. Our work's very motivation is connected to the interests of the five partner companies, and the bulk of the results were acquired through a workshop-survey with these companies. The workshops were held on-site with participants sampled using convenience sampling but included individuals with varying expertise and roles, e.g., project managers, senior architects, and developers. These results were complemented with literature from peer-reviewed articles about assets, artefacts, and technical debt, as well as the researchers' own empirical experiences and knowledge. The aggregated results were then synthesised to categorise and classify assets.

The paper is structured as follows: Section 2 provides the background and related work on the topic. Section 3 describes the research methodology, which separately covers the literature review and the industrial workshops. The results are presented in Section 4, together with the proposed Asset Management Taxonomy. Section 5 discussed the principal findings and the implications of the results. The threats to validity are also discussed and addressed in Section 5. And finally, Section 6 presents the conclusion and the continuation of the work together with the future directions.

2. Background and Related Work

Assets related to SIPS have been studied previously, for instance, from a managerial perspective where the term asset is used to discuss that the products in product lines that are developed from a set of core assets that might have built-in variation mechanisms [53]. In contrast to this work, our focus is instead on the inception, development, evolution, and maintenance of software related assets and does thereby not cover business- or market-related assets such as the work of Ampatzoglou et al. [3], Cicchetti et al. [15], Wohlin et al. [75], and Wolfram et al. [76].

2.1. Artefacts in Software Engineering

Describing how a software system is envisioned, built, and maintained is part of the Software Development Processes (SDP) [62]. The SDP prescribes the set of activities and roles to manipulate documents, i.e., software artefacts, namely source code, documentation, reports, and others [14]. The artefacts in software engineering are defined as i) "documentation of the results of development steps" [14]; ii) "a work product that is produced, modified, or used by a sequence of tasks that have value to a role" [49]; and iii) "a self-contained work result, having a context-specific purpose and constitutes a physical representation, a syntactic structure and a semantic content of said purpose, forming three levels of perception" [49]. Software artefacts are self-contained documentation and work products that are produced, modified, or used by a sequence of tasks that have value to a role [49].

The definition and organisation of software artefacts have a significant influence on software development. The descriptions and documentation in large-scale systems can grow exponentially and get "very large"; therefore, there is a need for structural organisation of software artefacts [14]. Artefacts defined by most of the SDPs are monolithic and unstructured [66]. The content of poorly structured artefacts is difficult to reuse, and the evolution of such monolithic artefacts is cumbersome [60]. Therefore, different SDPs present various models for presenting software artefacts, e.g., the Rational Unified Process (RUP) [32, 33]. There are many ways to classify and structure software artefacts based on well-known modeling concepts. Examples of such models are the work of Broy [14] and Silva et al. [60]. Moreover, there are ontologies and metamodels to classify artefacts in specific software development areas (e.g., Mendez et al. [50], Zhao et al. [78], and Constantopoulos and Doerr [17]).

The definitions of artefacts presented in the literature do not distinguish between the artefacts that have an inherent value for the development organisation[1] and are the artefacts that subject to degradation over time from the artefacts that do not have any value for the organisation. The value of each asset is a property that can characterise its degradation, i.e., if an asset degrades, it is still an asset, but its value for the organisation has degraded. Following this reasoning, an artefact that does not have value for the organisation cannot be classified as an asset and cannot be described with respect to its degradation.

2.2. Technical Debt and Assets

We have defined assets as artefacts with value for the organisation and, therefore, subject to degradation. Cunningham introduced the Technical Debt (TD) metaphor in 1992 to describe the compromises resulting from suboptimal decisions to achieve short-term benefits [18]. We assume that TD's focus is to identify the consequences of these suboptimal decisions on relevant assets [47], and that is why we study literature on this phenomenon intending to find relevant assets.

The TD metaphor has been extended and studied by many researchers [55]. It has been an interesting topic for both academia and industry, and it has grown from a metaphor to practice [36]. TD is currently recognised as one of the critical issues in the software development industry [8]. The activities that are performed to prevent, identify, monitor, measure, prioritise, and repay technical debt are called Technical Debt Management (TDM) activities [4, 26] and include, for example, identifying TD items in the code, visualising the evolution of TD, evaluating source code state, and calculating TD principal [55].

As the TD metaphor was extended to include different aspects of software development, various Technical Debt types were introduced [4], e.g., requirements debt, test debt, and documentation debt. The introduction of different types of TD has led researchers to attempt to classify the different

[1] Mendez et al. [49] define artefacts as having value for a role which is substantially different from having value for the organisation.

types and categories of TD. One of the earliest classifications of TD is the work of Tom et al. [67]. Other secondary and tertiary studies have been performed to summarise the current state of TD and TD types, e.g., by Lenarduzzi et al. [44], Rios et al. [55], and Li et al. [47].

The original TD definition focuses on source code, i.e., code, design, and architecture [4]. The organisational and social aspects of technical debt have not received the same amount of attention [55]. However, they might have been studied under different topics than TD. Moreover, TD types are often studied in isolation, not considering how they co-occur and the permeation to other TD types [4]. For example, when the code TD grows, a valid question would be whether and how it affects the test TD and the extent of such association's impact.

Moreover, the TD management activities have not been investigated thoroughly [4]. TD management activities have been investigated in several secondary studies with different perspectives, some focusing on tools, others on strategies, but there is still a lack of unified analysis aligning these different perspectives [55]. Lastly, TD is generally studied in the current state of the software and it is not studied with regards to the evolutionary aspects of TD, i.e., studying TD on a "snapshot" of a system is not enough [21, 54]. Therefore, a more appropriate approach to study TD is to study its evolution [13]. It is only by periodically monitoring TD that we can study the economic consequences of TD [19], determine the performance of the system in the future [20], and create methods and frameworks to react quickly to the accumulation of TD [45]. Based on research, but described slightly differently can be as follows. An asset has value for an organisation. If for any reason this value is degraded (e.g., code comments are not kept up to date due to time constraints, or an API description is not updated, etc.), you lessen the value of the said asset. If you identify this value degradation to be a problem, and you need to plan to invest resources into addressing the issue (e.g., updating code comments), this investment's cost can be seen as the repayment of the TD. Thus, TD is the delta between the current value of an asset and the value of said asset's preferred value (by the organisation).

2.3. Taxonomies in Software Engineering

Scientists and researchers have long used taxonomies as a tool to communicate knowledge as early as the eighteen century. One of the examples of early taxonomies is the work of Carl von Linné [48]. Taxonomies are mainly created and used to communicate knowledge, provide a common vocabulary, and help structure and advance knowledge in the field [24, 40, 70]. Taxonomies can be developed in one of two approaches; top-down, also referred to as enumerative, and bottom-up, also referred to as analytico-synthetic [12]. The taxonomies that are created using the top-down method use the existing knowledge structures and categories with established definitions. In contrast, the taxonomies that use the bottom-up approach are created using the available data such as experts' knowledge and literature, enabling them to enrich the existing taxonomies by adding new categories and classifications [69].

Software Engineering (SE) is continually evolving and becoming one of the principal fields of study with many subareas. Therefore, the researchers of the field are required to create and update the taxonomies and ontologies to help mature, extend, and evolve SE knowledge [70]. The Guide to the Software Engineering Body of Knowledge (SWEBOK) can be considered as a taxonomy that classifies software engineering discipline and its body of knowledge in a structured way [11]. Software engineering knowledge areas are defined in SWEBOK, and they can be used as a structured way of communication in the discipline. Other examples of taxonomies in software engineering are the work of Glass et al. [25] and Blum [10]. Specialised taxonomies with narrower scopes are also popular in the field. These taxonomies are focused on specific sub-fields of software engineering such as Taxonomy of IoT Client Architecture [65], Taxonomy of Requirement Change [58], Taxonomy of Architecture Microservices [23], Taxonomy of Global Software Engineering [61], and Taxonomy of Variability Realisation Techniques [64] to name a few.

This paper presents a taxonomy of assets in the inception, planning, development, evolution, and maintenance of a SIPS. The taxonomy is built using a hybrid method, i.e., the combination of top-down and bottom-up. The details of the taxonomy creation are presented in Section 3.4. The purpose of our taxonomy is to cluster.

2.4. Summary of the Gaps

In this paper, we introduce assets in software development and software engineering. We use the concept of artefacts and their intentional degradation from the research in the field of Technical Debt as a starting point for this work, which we expand to include various types of valuable artefacts as Assets.

In particular, we aim to address the following challenges and gaps:

- Identifying and distinguishing between the artefacts with inherent value (assets) with those without.
- Identifying assets by considering every aspect of software development. For example, Environment-and-Infrastructure-, Development-Process-, Ways-of-Working-, and Organisation-related aspects [4, 55] that have not received much attention in TD.
- Laying the groundwork for capturing the evolutionary aspect of assets and their degradation. The TD metaphor generally applies to the current state of the product and dismisses the assets' evolutionary aspect [21, 54]. As described in Section 2.2, studying TD with regards to its evolution is needed to understand the economic consequences of TD [19], the quality of the system in the future such as system performance [20], and the creation of TD management frameworks [45].

Based on the asset's definition in [77], we identify the artefacts that adhere to this definition and which of these are common in the industry. There are adjacent and similar taxonomies that cover subsets and part of this work. However, their purposes are different than what are aiming to achieve. To the best of our knowledge, there is no ontology or taxonomy that we can use to describe and classify assets and their interrelations.

3. Research Overview

This section presents the research methodology and design of the study. The study is divided into two parts; a literature review and an empirical study of industrial cases. In the following subsections, we first cover how we performed the literature review (Section 3.2). We explain how the data was gathered, coded, and analysed. We describe the process of studying company cases, the data collection, and the analysis (Section 3.3). Later, we describe how the taxonomy was created based on the collected data (Section 3.4). The steps taken to internally validate the taxonomy by the researchers involved in the industrial workshops[2] are presented (Section 3.5).

To make the research objective more concrete, it has been broken down into the following research question:

RQ : *What assets are important for organisations during the inception, planning, development, evolution, and maintenance of SIPS?*

We aim to provide a systematically constructed and extendable taxonomy to accommodate the set of terms and concepts used in both academia and industrial practice as it pertains to SIPS.

We performed a literature review to capture the classifications and definitions of various assets studied and presented in research venues (top-down method). Included are systematic literature reviews, systematic mapping studies, and tertiary studies. The details of the literature review procedure are presented in Section 3.2.

To study state-of-practice, we performed industrial workshops over multiple companies to find evidence on how assets are defined and used. The five cases were selected using convenience sampling as companies involved in an ongoing research project that is interested in addressing asset degradation challenges. We organised several workshops with the industrial partners to investigate and discuss the topics with experts in each area. The reports and results from the workshops were then coded and used for the construction of the taxonomy. We used the bottom-up method for updating the existing structure that we obtained from the literature review. This approach enabled us to enrich the existing structure from the literature. The details of the procedure of the industrial workshops are presented in Section 3.3.

Usman et al. [70] present a revised method for the development of taxonomies in software engineering. The revised method includes four phases of *Planning, Identification and Extraction, Design and Construction,* and *Validation*. The asset management taxonomy was created by following the mentioned method.

Phase 1: during which the taxonomy's context and objective (software engineering knowledge area, objective, subject matter, classification structure, and information sources) were decided. This phase is corresponding to the *planning* phase of the method (see Section 3.1)

Phase 2: during which the raw data for creating the taxonomy was collected. This phase is corresponding to the *identification and extraction* phase of the method (see Section 3.2 and Section 3.3)

Phase 3: during which the taxonomy categories were extracted, and the relations between the items (i.e., the nodes in the taxonomy tree) were identified. This phase is corresponding to the *design and construction* phase of the method (see Section 3.4)

Phase 4: during which the taxonomy was internally validated by utility demonstration. The utility of the taxonomy was demonstrated by expert opinion (i.e., researchers). This phase is corresponding to the *validation* phase of the method (see Section 3.5)

3.1. Planning Taxonomy Creation

The complexity of developing software-intensive products and services makes it challenging to identify and maintain the assets [77]. In such cases, the need for an organised, well-structured body of knowledge is crucial. We discussed the concept and definition of assets in our previous work [77] and planned to create a taxonomy of assets utilising the knowledge from academia and the industry to capture the body of knowledge in an organised structure.

3.2. Literature Review

To investigate the terminology used in the area of assets and asset management, we reviewed research papers that classify assets and provide definitions of the term.

The assets (i.e., artefacts that we are interested in) are subject to technical debt which makes the research on TD a good input area to find assets. However, we do not delimit ourselves to the artefacts that are discussed in TD. We chose to do a literature review on the topic of technical debt and the classifications provided by the TD literature (i.e., TD types) because the TD metaphor is a subset of asset management (see Figure 1). Technical Debt is part of Asset Degradation defined in [77]. Moreover, the TD literature is concerned with different aspects of software development. The TD metaphor has been used as an umbrella term to include all the assets involved during the development. TD has received a growing interest in academia and the industry. And the TD topic has been studied in various venues and research for a considerable time.

We began by collecting the start set of the papers using *"technical debt" AND ("systematic literature review" OR "systematic mapping study" OR "tertiary study")* as the search string in Google Scholar. We selected the articles that presented a classification for TD. Finally, we completed

[2] Industrial workshops refer to the workshops where the data was collected.

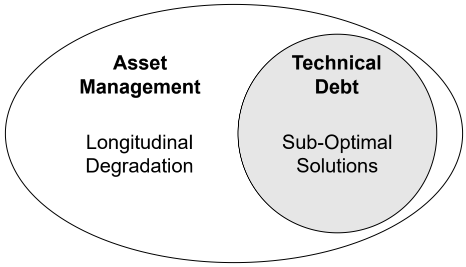

Figure 1: Venn diagram depicting how Asset Management and Technical Debt are related in their context.

the set by following the snowballing iteration guidelines provided by Wohlin [73].

The execution process included the following steps as illustrated in Figure 2. The results of this process are presented in Section 4.1.

- **Step 1:** Collection of the start set of relevant articles (seed papers) including SLRs, SMSs, and Tertiary studies on technical debt.
- **Step 2:** Evaluate the papers - start set in the first iteration - for inclusion/exclusion based on the criteria, i.e., papers that presented any classification of TD and artefacts affected by TD.
- **Step 3:** The snowballing procedure for identifying additional secondary and tertiary studies on TD that satisfy our inclusion/exclusion criteria:
  - **Step 3.1:** Backward snowballing by looking at the references of the papers.
  - **Step 3.2:** Forward snowballing by looking at the papers that cite the papers.
- **Step 4:** Extracting different *types of assets* and *assets* together with their respective definitions from the selected articles.
- **Step 5:** Synthesising the definitions of the *types of assets* and *assets* provided by the selected articles.
- **Step 6:** Creating the matrix of *types of assets* and *assets* based on technical debt classifications defined by the selected articles.

3.3. Industrial Workshops

To collect data, we performed workshops with industrial partners that develop and maintain software-intensive products and services according to the process presented in Figure 3.

3.3.1. Case Company Characterisation

We have collected the data for this study collaborating with five companies that work construction machinery industry, communication technology industry, and banking and financial services. The research partner companies are Ericsson (telecommunication & ICT), Fortnox (Finance), Time People Group / Qtema (Consultancy), and Volvo CE (Automotive Industry). The companies were selected by convenience and availability. The partner companies are mature in their development practices and have well-established, successful products. They are interested in continuously improving their products and development life-cycles, which turns into their willingness on participating in studies like this. All the collaborating companies work on developing software-intensive products and services and are involved in an ongoing research project[3]. The details of the case companies are presented in Table 1. Note that the order of the companies in Table 1 does not correspond with the order of workshops (workshops IDs) in Table 4, which has been shuffled to preserve confidentiality.

3.3.2. Workshop Procedure

The workshops include six steps (see Figure 3):

- ***IW1.*** Workshop participants introducing themselves.
- ***IW2.*** One of the moderating researchers presenting the topic.
- ***IW3.*** Workshop participants discussing the topic, providing insight into their views/experiences with TD and document them in notes.
- ***IW4.*** Workshop participants discussing assets and asset management in detail after a second presentation of the concept.
- ***IW5.*** Workshop participants discussing what they wrote before as technical debt examples and rethinking them in terms of assets, asset degradation, and asset management.
- ***IW6.*** A closing discussion and focus groups.

Each workshop starts with participants introducing themselves with background information about their work, including their current role in the organisation (step IW1). One of the moderating researchers then presents the workshop's agenda and covers the importance of the topic and the growing interest in value creation and waste reduction both in academia and in the industry (step IW2).

After the initial introduction of the topic by the moderating researchers, the participants are divided into groups. They are asked to list and discuss the challenges with their ways of working (while considering varying aspects of technical debt), i.e., the problems they know or have encountered or experienced (step IW3). After the time is up, the notes are read, discussed, and abstracted to a more general description and later put on a whiteboard. The connections between the items on the board are identified and marked down with a marker.

After the second presentation, i.e., introducing the participants with the concepts related to Asset Management (AM) and Asset Degradation (AD) [77] (step IW4), participants

[3] See www.rethought.se.

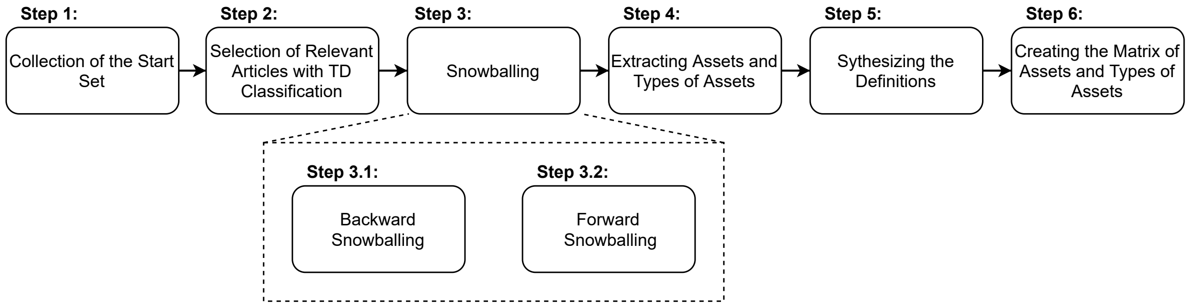

Figure 2: The literature review execution process.

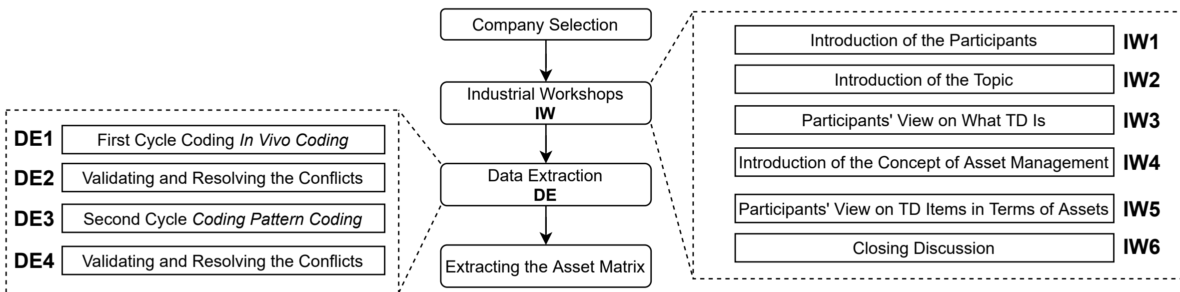

Figure 3: The execution process for industrial cases.

add new items to the previous notes on the board. Participants then refine the items from the board for the rest of the workshop (step IW5). The workshop ends with a closing discussion on the topic and the items (step IW6).

In the context of this research, we have moved the focus from the traditional technical debt metaphor to asset degradation. In this framework, we talk about asset degradation as the deviation of an asset from its representation. That way, we can focus, potentially, on any type of asset and its representation. This framework provides us with a broader, holistic view that allows us to study how an asset's degradation (e.g., requirements) might introduce degradation in other assets (e.g., code or test cases) [77].

The written notes from the participants, the minutes of the workshops, as well as the final reports that were sent to companies after the workshops were used as raw data. The researchers' minutes were written during each session and were then aggregated and summarised in a report sent to the companies. The notes were used for coding and later to extract *types of assets* and explicit *assets*. The details of the data extraction and taxonomy creation are described in Section 3.4.

3.3.3. Data Extraction

Two sources of raw data were gathered from the industrial workshops; written notes from workshop participants and notes from moderating researchers specifically assigned to scribe each session's discussions. To create the matrix of assets from industrial insights, we use the hybrid method of coding, as Saldaña [59] suggested. The coding is divided into two main cycles: First Cycle Coding and Second Cycle Coding.

First Cycle Coding of the raw data happens in the initial stage of coding. The raw data, which can be a clause, a sentence, a compound sentence, or a paragraph, is labelled based on the semantic content and the context in which it was discussed during the workshop. We have used the *in vivo coding* method to label the raw data in the first cycle. In vivo coding is suitable for labelling raw data in the first cycle coding. It prioritises the interviewees' / participants' opinions [59]. It adheres to the "verbatim principle, using terms and concepts drawn from the words of the participants themselves. By doing so [the researchers] are more likely to capture the meanings inherent to people's experiences." [63, p. 140] It is commonly used in empirical and practitioner research. [63, 16, 22]

The coding was done by two researchers independently (step DE1, see Figure 3). The labels were then compared to validate the labels and to identify conflicting cases. A third researcher helped to resolve the conflicts by discussing the labels with the two initial researchers (step DE2, see Figure 3).

Second Cycle Coding is done primarily to categorise, theorise, conceptualise, or reorganise the coded data from the first cycle coding. We have used Pattern Coding [59]

Table 1
Case company details. The table is ordered alphabetically based on the name of the companies, and does not correspond to the order in Table 4.

Company	Domain	Investigated Site	Enterprise Size	Participants' Roles
Ericsson	Telecommunication & ICT	Karlskrona, Sweden	Large	Senior System Architect Corporate Senior Strategic Expert Operations & Testing
Fortnox	Finance	Växjö, Sweden	Large	Head of Development Product Owners Development Managers System Architect Testing
Qtema†	Consultancy	Stockholm, Sweden	SME	Chairman of the Board Requirements Analyst Sales Manager Project Manager IT Administration Manager
Time People Group†	Consultancy	Stockholm, Sweden	SME	Data Consultant Project Manager Consultant Senior Agile Coach IT Project Manager Team Leader Chief Exevutive Officer (CEO) Consultant Test Leader
Vovlo CE	Construction Machinery	Gothenburg, Sweden	Large	Enterprise Architects Solution Architect Business Information Architect

† Time People Group and Qtema participated in the same workshop.

as the second cycle coding method. According to Miles et al. [51, p. 86], pattern coding is used in cases where: (i) the researchers aim to turn larger amounts of data into smaller analytical units. (ii) the researchers aim to identify themes from the data. (iii) the researchers aim to perform cross-case analysis on common themes from the data gathered by studying multiple cases.

Similar to the first cycle coding process, pattern coding was done by two researchers independently (step DE3, see Figure 3). The results were compared to validate the classifications and to identify conflicting cases. A third researcher resolved the conflicting cases in a discussion session with the two researchers (step DE4, see Figure 3). The results of the insights gathered from industrial workshops are presented in Section 4.2.

3.4. Taxonomy Creation

To describe a precise syntax and the semantics of the different concepts used for the taxonomy creation, we created a metamodel (presented in Figure 4). The metamodel summarises and identifies the metaclasses, i.e., the characteristics of assets, their definitions, and their relation. The metamodel illustrates the structural relationships between the metaclasses. The metaclasses presented in the metamodel are:

- The *"AssetComponent"* metaclass is the container metaclass for the items in the model.
- The "TypeOfAsset" metaclass represents the hierarchical classification of assets. The items belonging to this metaclass can be further broken down into subclassifications representing various groups of assets. The types of assets are containers for the assets. Asset types are identified from the state-of-the-art (i.e., existing academic literature), state-of-practice (i.e., the industrial insights gathered through the industrial workshops), or the identified by researchers.
- The *"Asset"* metaclass represents atomic and measurable assets. Each asset belongs to a type of asset. Assets are identified from the state-of-the-art (i.e., existing academic literature), state-of-practice (i.e., the industrial insights gathered through the industrial workshops), or identified by researchers.
- The *"Reference"* metaclass represents the references from which each asset or type of asset has been identified. Reference can originate from academic literature (the literature review) or industrial insights (gathered from industrial workshops). References can be mapped to individual *assets/type of assets* or multiple

assets/type of assets.

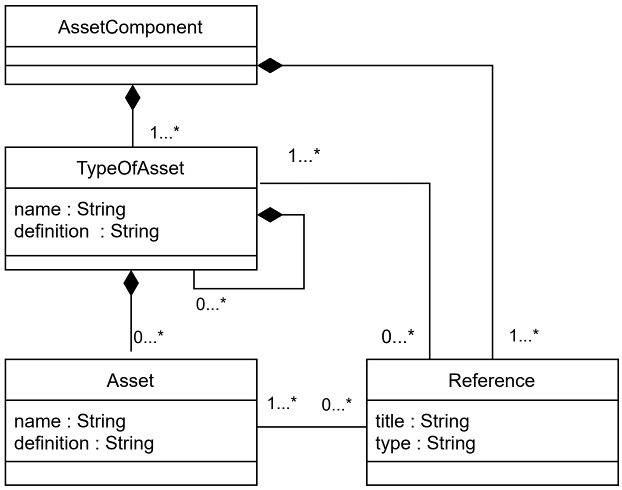

Figure 4: Asset Management Metamodel.

The creation of the taxonomy included three steps. First, we created an asset matrix based on the technical debt types (top-down approach) with items that we extracted from the literature review. We created the matrix based on the literature's definitions, i.e., by synthesising the definitions to identify similarities, differences, and hierarchies of identified items. We have grouped the definitions provided by the literature based on their semantic meaning.

In the second step, we utilised the extracted assets from industrial workshops (Section 3.3.3) to create a second asset matrix (bottom-up approach). Like the previous step, we used the definitions of the assets and their types and the participants' statements from the workshops.

By combining the two matrices in the last step, we created the asset management tree. To complete the tree, we added some nodes based on the researchers' expertise that we perceive were missing nodes and leaves. We mention such cases as *Author Defined Assets* (ADA) when presenting the results.

The process of adding ADA started with researchers suggesting assets that should be included in the taxonomy. These suggested assets were brought up in internal workshops [4] where all the researchers discussed, reflected, improved, and added / removed the suggested assets. *User Stories*, as an example, were suggested by one of the researchers to be considered as assets during one of the internal workshops. The discussion was regarding (i) whether or not *User Stories* are assets, (ii) if they fit in the taxonomy according to the definition, (iii) where they belong in the tree, (iv) what they represent, (v) and how they degrade. After the discussions, the researchers decided that *User Stories* belong to *[AM1] - [AM1.1] Functional-Requirements-Related Assets* in the taxonomy tree.

[4] Internal workshops refer to the workshops where the researchers discussed the taxonomy.

The assets included in the taxonomy should adhere to the definition of an asset presented in [77], thereby also the following inclusion criteria:

- Assets are artefacts that have inherent value for the organisation and are subject to degradation, i.e., the value of the asset can degrade (lessen).
- Assets are persistent and not inherently transient, i.e., we do not consider intermediate entities as assets. In our definition, artefacts that are created with a particular purpose and then immediately transformed to other artefacts are not assets.

3.5. Internal Validation

The taxonomy was created by the authors after the data collection was complete. We conducted internal workshops to validate the taxonomy and its structure with the seven researchers who were involved during the industrial workshops.

Step 1: The taxonomy was sent to all the researchers to study and prepare for the internal workshop.

Step 2: An internal workshop was held where the first and second authors presented the taxonomy and the researchers provided their feedback. The discussions were on both the structure of the taxonomy and on the assets.

Step 3: The taxonomy was updated based on the discussion and the feedback provided during the internal workshops.

Step 4: The authors held individual meetings with each researcher to further discuss the latest updates on the taxonomy and delve into the areas of expertise of each researcher.

Step 5: The taxonomy was updated based on the individual meetings and presented to the researchers for the final approval.

4. Results

This section presents the results of the literature review, then the results of the workshops. Finally, we will present the Asset Management Taxonomy based on both.

4.1. Input from the Literature

The final list of papers included nine articles presented in Table 2. To create the assets matrix, we extracted the types of technical debt from each article. For example, in paper *P8* [44], we refer to Table 1 on page 4 of the article, where the authors summarise different types of technical debt and their respective definitions. The authors define Requirements TD as "the distance between the optimal requirements specifications and the actual system implementation, under domain assumptions and constraints" [44]. Requirements is also mentioned as a TD item in *P3*, *P4*, *P5*, and *P6*. Table 3 presents the asset matrix based on the types of technical debt. The codes of the papers are used as a reference throughout this paper.

Looking at Table 3 we can see that there are fewer categories in the earlier studies (*P1* and *P2*). The papers that follow these studies break down the bigger categories into

Table 2
The articles gathered for the literature review during the snowballing process.

Code	Title	Seed Paper	Backward Snowballing	Forward Snowballing
P1	A Consolidated Understanding of Technical Debt [67]		x	
P2	An Exploration of Technical Debt [68]		x	
P3	Towards an Ontology of Terms on Technical Debt [2]	x		
P4	A Systematic Mapping Study on Technical Debt and Its Management [47]	x		
P5	Identification and Management of Technical Debt: A Systematic Mapping Study [1]	x		
P6	Managing Architectural Technical Debt: A Unified Model and Systematic Literature Review [9]	x		
P7	A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners [55]	x		
P8	Technical Debt Prioritisation: State of the Art. A Systematic Literature Review [44]			x
P9	Investigate, identify and estimate the technical debt: a systematic mapping study [6]			x

more specific categories. For example, *Architecture and Design* are put into one category in *P1* and *P2* and later, they are broken down to their own categories. It is important to mention that *P7* has fewer categories since the study is on the specific topic of *Architectural Technical Debt*.

4.2. Input from the Workshops

There were a total number of four workshops, each held with participation of different companies. The workshops' procedure stayed the same while the closing discussion of each workshop was on the topic of interest for the workshop participants (the stakeholders). The topics included, but were not limited to, *Lack of Knowledge/Competence*, *Architecture Artefact Lifecycle*, *Business Models for Products*, and *Backlog Update Issues/Backlog Size*.

The data extraction process was done, as described in Section 3.3.3. Two researchers used the in vivo coding method to label 386 statements during the first cycle coding. After matching and validating the labels, 14 cases of conflicting labels were identified. The conflicting labels were resolved during a discussion session with a third researcher. The researchers agreed on the new labels for the conflicting cases during the discussion.

After solving the conflicting cases of in vivo coding, the researchers continued with pattern coding [59] of the first cycle coding results. The two researchers classified the assets based on in vivo labels. Table 4 presents the asset matrix based on the results of the second cycle coding.

The workshops presented in Table 4 are chronological, i.e., *WS1* was the first workshop. Though the workshops were executed using the same procedure, we only extracted relevant statements. Therefore, *WS1* has fewer entries in Table 4. That is why we observe only two assets/types of assets in the first column, and this might be due to the profiles or experience of the participants.

Examining Table 4, we can observe that assets are mentioned more often than types of assets in the industrial workshops whereas types of assets are more frequent in the literature review (see Table 3). Finally, assets that are related to *Operations*, *Management*, and *Organisational Management* were highlighted more in the industrial workshops than the literature review.

It appears that the assets that are mentioned frequently are:

- **Easier to contextualise**: It is easier for the stakeholders to identify such assets in the software product context. For example, the data that the company acquires from the operation of the product, i.e., *Application Data*, can be used as input to improve the product.
- **More tangible**: The assets that have been studied and discussed before, and the concept is not foreign anymore. For example, every software company, one way or another, has a *Product Backlog* with specific characteristics which is familiar for all the people involved in the development of the software-intensive product.
- **Easier to measure**: There are already existing metrics used to measure the state of such assets. For example, there are many metrics available to measure *Source Code*, such as LOC and Cyclomatic Complexity.
- **Used universally**: The assets that are defined in the same way across different organisations and academia, meaning that they are not organisation-specific. For example, the *software's architecture (Code Structure)* is a universal and inherent aspect of any software-intensive product.

4.3. The Asset Management Taxonomy

Using the key concepts extracted from the labelled data (presented in Table 3 and Table 4), we build the taxonomy of assets. The taxonomy contains the terminology identified through the literature review and the industrial workshops. The terms included in the taxonomy are presented in a tree

Table 3
Asset matrix from technical debt literature.

P1 2012	**P2 2013**	**P3 2014**	**P4 2015**	**P5 2016**	**P6 2018**	**P7 2018**	**P8 2019**	**P9 2020**	**Emerging Category(ies)**	**Phase, during which the artefact is produced**
Features		Requirements	Requirements	Requirements Usability	Requirements Usability		Requirements	Requirements Usability	*Product Requirements* *Quality Requirements*	Requirements
Design\ Architecture Documentation	Design and Architecture Documentation	Architecture Design Documentation Design Specifications	Architecture Design Documentation	Architecture Design Documentation	Architecture Design Documentation	Architecture	Architecture Design Documentation Architectural Documentation	Architecture Design Documentation	*Architecture Design Decisions Documentation* *Product Documentation* *Architectural Documentation*	Design
Code	Code	Code Build Service	Code Build Versioning	Code Build Service Versioning	Code Build Service Versioning	Code	Code Build Versioning	Code Build Service Versioning	*Source Code* *Build Documentation* *Web Services* *Versioning*	Development
Testing Defects/ Known	Testing	Test Defects Test Automation	Test Defect	Test Defect Test Automation Test Case Documentation	Test Defect Test Automation		Test Defect	Test Defect Test Automation	*Functional Tests* *Test Automation* *Test Documentation*	Verification and Validation
Infrastructure	Environment Hardware Infrastructure Operational Processes	Infrastructure	Infrastructure	Infrastructure	Infrastructure		Infrastructure	Infrastructure	*Environment and Infrastructure* *Environment and Infrastructure* *Operations*	Operations
		Process		Process Documentation Specifications	Process			Process	*Process Managmenet* *Documentation* *Internal Rules and Specifications*	Management

Color Guide: Assets | Types of Assets | Temporary Artefacts

Table 4
Asset matrix from industrial workshops.

WS1	WS2	WS3	WS4	Emerging Category(ies)	Phase, during which the artefact is produced
Contradictory Requirements	Requirements	Requirements	Requirements	*Product Requirements*	Requirements
	Documentation Architectural Documents Architecture	Architectural Models Software Structure	Documentation Architectural Documentation Architecture	*Documentation* *Product Documentation* *Architectural Documentation* Architectural (Source Code)	Design
Dangerous Code	Code APIs Libraries	Code API Versions Third Party Products	Code	Source Code *APIs* Libraries\ External Libraries	Development
	Test Cases Automated Tests Bug Reports	Tests	Test Cases	*Test Cases* *Test Automation Scripts*	Verification and Validation
	Kubernets	Containers\Kubernets	Application Data	*Application Data* *Tools*	Operations
	Ways of Working Coding Standards Product Roadmap Backlog	Ways of Working Coding Standards Architectural Rules Product Management	Documentation about Ways of Working Documentation Standards	*Documentation about Ways of Working* *Coding Standards* *Architectural Internal Standards* *Documentation Internal Rules\Standards* *Product Management* *Product Backlog*	Management
	Organisation's Roadmap	Business Models	Holistic Strategy Organisation's Structure	*Organisation's Strategy* *Organisation's Structure* *Business Models*	Organisational Management

Color Guide: Assets | Types of Assets | Temporary Artefacts

(graph). The nodes represent the assets (the leaf nodes) and the types/categories of assets (non-leaf nodes).

The tree presented in Figure 5 contains the types of assets (The full tree is presented in Appendix A). Note that the nodes in the tree in Figure 5 are mapped to represent their source. For example, a node can be assigned with *[P1]* as a reference for an article in the literature review. Similarly, *[WS1]* as a reference for an asset coming from industrial workshops[5]. And finally, Author Defined Assets (*[ADA]*) which are assets included in the taxonomy by the researchers although it was not coming from industrial workshops or from the TD literature that we have covered but is supported by SE literature.

The following example clarifies the distinction between *Temporary Artefact* (grey cells presented in Table 3 and Table 4) and *Assets*:
An API description used by developers as a reference has value in the development effort. If changes are made (new decisions, new ways to adhere to components, etc.), but the API description is not updated to reflect this, the utility (value) of the API description becomes lower (the asset degrades). On the other hand, an automatically generated bug report is not seen as an asset, as it is transient or intermediate. It is generally created as a work product used once to be "transformed" into "change requests" or other management artefacts. Once transformed, it is discarded to be replaced with a new bug report in subsequent test runs.

The process of combining the asset matrices from the literature review, the input from the industrial workshop, and completing the tree with *Author Defined Assets* (ADA) resulted in the taxonomy containing 24 asset types and the total of 57 assets. In the following subsections, we will first present the types of assets and then describe the eight major types of assets labelled *AM1-AM8*. We include the definitions of each asset type together with their corresponding assets. After following the process described in Section 3.4, the following major types of assets were extracted:

Product-Requirements-Related Assets (AM1) refer to the types of assets and assets concerned with software requirements, including the elicitation, analysis, specification, validation, and management of requirements during the life cycle of the software product.

Product-Representation-Related Assets (AM2) refer to the types of assets and assets concerned with system-and architectural design and any documentation related to these artefacts.

Development-Related Assets (AM3) refer to the types of assets and assets concerned with the development of the software product, including the code, build, versioning, and arte-

[5] The IDs on the workshops on Table 4 have been obfuscated to preserve anonymity, and has no relationship with the order of companies shown in Table 1.

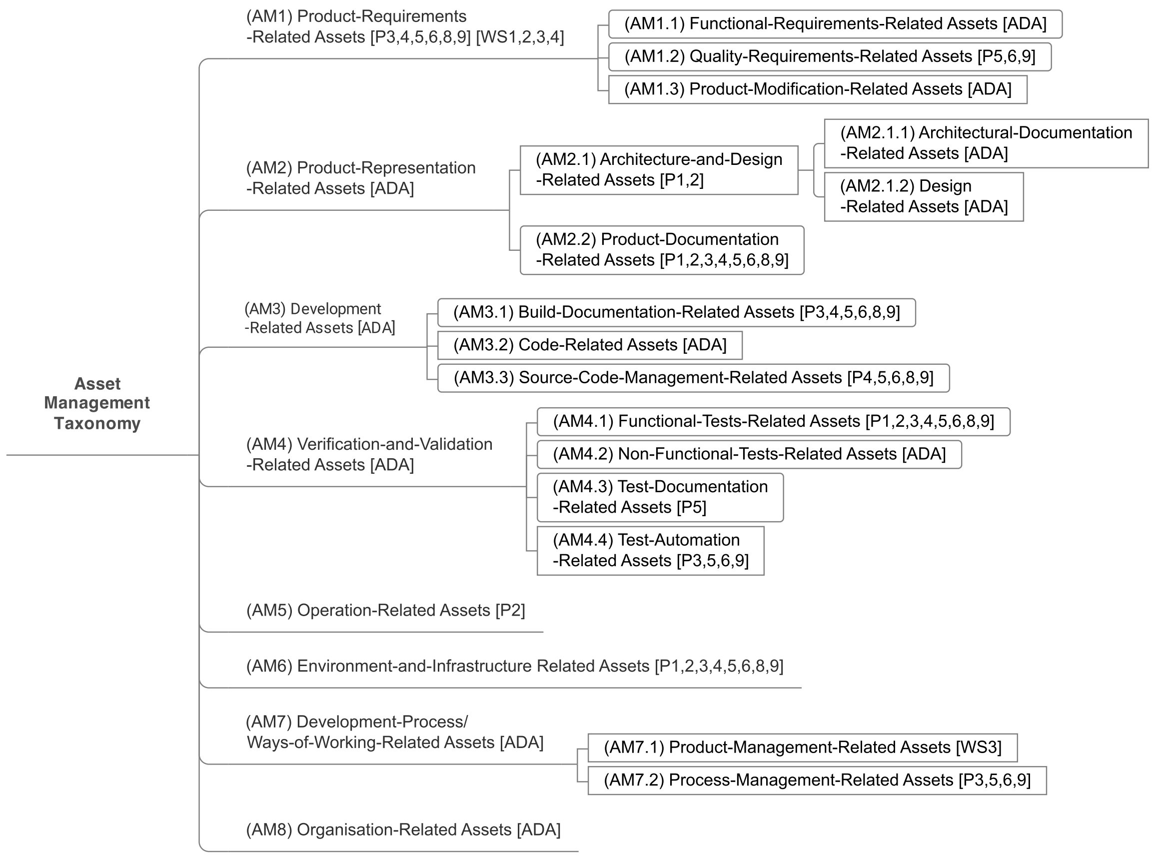

Figure 5: The Asset Management Taxonomy. The tree contains only the types of assets. The full tree is presented in Appendix A.

facts related to them.

Verification-and-Validation-Related Assets (AM4) refer to the types of assets (including sub-types) and assets concerned with software testing and quality assurance, and the output provided by such sub-type assets that help the stakeholders investigate the quality of the software product.

Operations-Related Assets (AM5) refer to the types of assets and assets concerned with the data produced from operational activities.

Environment-and-Infrastructure-Related Assets (AM6) refers to the types of assets and assets concerned with the development environment, the infrastructure, and the tools and artefacts (including support applications) that facilitate the development process.

Development-Process/Ways-of-Working- Related Assets (AM7) refer to the types of assets and assets concerned with all the interrelated processes and procedures that transform inputs into outputs during the development process.

Organisation-Related Assets (AM8) refer to the types of assets and assets concerned with the organisation itself, such as team constellation, team collaborations, and organisational governance.

4.3.1. Product-Requirements-Related Assets (AM1)

Product-Requirements-Related Assets include the following three types of assets (see Figure 6). Table 5 presents product-requirements-related assets, their properties, and definitions.

Functional-Requirement-Related Assets (AM1.1) refer to the assets related to the functions that the software shall provide and that can be tested [11]. We have observed the following assets belonging to this type: *Feature-Related Backlog Items*, *User Stories*, and *Use Cases*.

Quality-Requirement-Related Assets (AM1.2) refer to the assets related to non-functional requirements that act to constrain the solution [11]. We have observed the following assets belonging to this type: *System Requirements*, *User Interface Designs*, *Quality Scenarios* (i.e., the -ilities), and *User Experience Requirements*.

Product-Modification-Related Assets (AM1.3) refer to assets that mandate a change of the system and, but not necessarily, the requirements. *Change Requests* is an asset we observed belonging to this type.

4.3.2. Product-Representation-Related Assets (AM2)

Product-Representation-Related Assets include the following two types of assets (see Figure 7). Table 6 presents product-representation-related assets, their properties, and

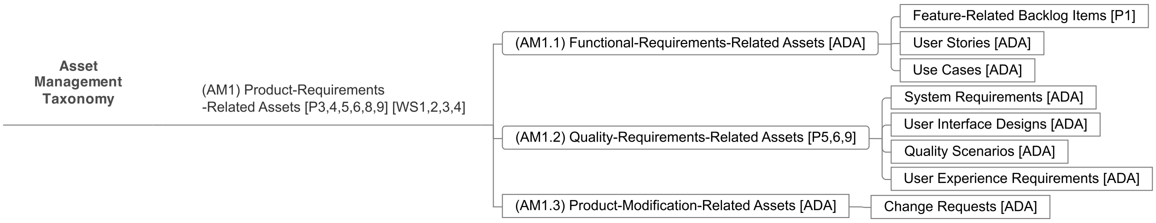

Figure 6: Product-Requirements-Related Assets Subtree.

Table 5
Product-Requirements-Related Assets' Definitions.

Asset	AM type	Definition
Feature-Related Backlog Items	AM1.1	Feature-Related Backlog Items are the results of refining and breaking down the user stories to create executable tasks [52].
User Stories	AM1.1	User Stories are, according to the agile development paradigm, a way to specify the features of the software that is being developed [52].
Use Cases	AM1.1	Use Cases are lists of actions or events that describe how a user will achieve a goal in a system [34].
System Requirements	AM1.2	"System Requirements are the requirements for the system as a whole. System Requirements [...] encompass user requirements, requirements of other stakeholders (such as regulatory authorities), and requirements without an identifiable human source." [11].
User Interface Designs	AM1.2	"User Interface Design is an essential part of the software design process. User interface design should ensure that interaction between the human and the machine provides for effective operation and control of the machine. For software to achieve its full potential, the user interface should be designed to match the skills, experience, and expectations of its anticipated users." [11].
Quality Scenarios (The -ilities)	AM1.2	"A quality attribute (QA) is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders." [5] A quality scenario is a way of stating a requirement in an unambiguous and testable manner [5].
User Experience Requirements	AM1.2	User Experience Requirements "are considered key quality determinants of any product, system or service intended for human use, which in turn can be considered as product, system or service success or failure indicators and improve user loyalty." [41, 38].
Change Requests	AM1.3	Change Requests are the modifications to the software product that are not coming from the requirements analysis of the product.

definitions.
Architecture-and-Design-Related Assets (AM2.1) refer to the assets that are used to design, communicate, represent, maintain, and evolve the software product, which is divided into:

- *Architectural-Documentation-Related Assets (AM2.1.1)* refer to the assets used to design, communicate, represent, maintain, and evolve the architectural representation of a software product. We have observed the following assets belonging to this type: *Architectural Models* and *Architectural Documentation*.

- *Design-Related Assets (AM2.1.2)* refer to the assets that belong to the design and design artefacts used during the development process. We have observed the following assets belonging to this type: *Design Decisions Documentation* and *System Designs*.

Product-Documentation-Related Assets (AM2.2) refer to the assets that belong to the product documentation and the process of creating such documentation. We have observed the following assets belonging to this type: *Documentation Automation Scripts* and *Product Documentation*.

4.3.3. *Development-Related Assets (AM3)*

Development-Related Assets include the following three types of assets (see Figure 8). Table 7 presents the development-related assets, their properties, and definitions.

Build-Documentation-Related Assets (AM3.1) refer to the assets related to the build system itself, the build environment, and the build process. We have observed the following assets belonging to this type: *Build Plans*, *Build Results*, and *Build Scripts*.
Code-Related Assets (AM3.2) refer to the assets that are related to the source code. We have observed the following assets belonging to this type: *Source Code*, *Code Comments*, *APIs*, *Architecture (Code Structure)* —i.e., a set of structures that can be used to reason about the system including the elements, relations among them, and their properties [5]—, and *Libraries/External Libraries*.
Source-Code-Management-Related Assets (AM3.3) refer to the assets related to managing the source code such as versioning and problems in code versioning and burndown charts. *Versioning Comments* is an asset we observed belonging to this type.

4.3.4. *Verification-and-Validation-Related Assets (AM4)*

Verification-and-Validation-Related Assets include the following four types of assets (see Figure 9). Table 8 presents verification-and-validation-related assets, their properties, and definitions.
Functional-Tests-Related Assets (AM4.1) refer to the assets related to testing the functionality of the system, its related features, and how they work together. We have observed the

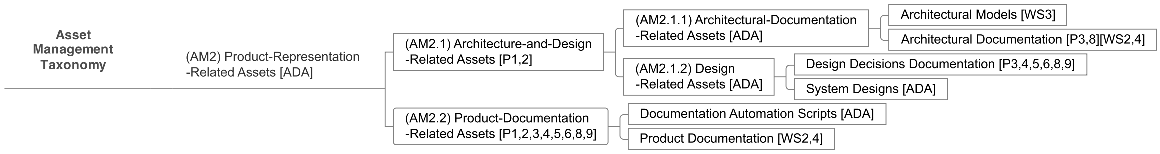

Figure 7: Product-Representation-Related Assets Subtree.

Table 6
Product-Representation-Related Assets' Definitions.

Asset	AM type	Definition
Architectural Models	AM2.1.1	Architecture Models are partial abstractions of systems, they capture different properties of the system [37]. "Architecture modeling involves identifying the characteristics of the system and expressing it as models so that the system can be understood. Architecture models allow visualisation of information about the system represented by the model." [39]
Architectural Documentation	AM2.1.1	Architectural Documentation are the representations of the decisions made to construct the architecture of the software [37].
Design Decisions Documentation	AM2.1.2	Design Decisions Documentation are the results of the design decisions that architects create and document during the architectural design process [11].
System Designs	AM2.1.2	System Designs are the processes of defining elements of a system. These elements are specified in the requirements and are extracted to create modules, architecture, components and their interfaces and data for a system.
Documentation Automation Scripts	AM2.2	Documentation Automation Scripts are the scripts that generate documentation based on the state of the source code.
Product Documentation	AM2.2	Product Documentation are the operational guidelines (such as *user manuals* and *installation guides*) for when the product is in use.

following assets belonging to this type: *Unit Tests*, *Integration Tests*, *System Tests*, and *Acceptance Tests*.

Non-Functional-Test-Related Assets (AM4.2) refer to the assets related to testing the quality attributes of the system and whether they satisfy the business goals and requirements. We have observed the following assets belonging to this type: *Non-Functional Test Cases* and *Non-Functional Test Results*.

Test-Documentation-Related Assets (AM4.3) refer to the assets related to documenting the testing process. *Test Plans* is an asset we observed belonging to this type.

Test-Automation-Related Assets (AM4.4) refer to the assets that are utilised for automated testing of the system. We have observed the following assets belonging to this type: *Test Automation Scripts* and *Test Automation (Real/Synthetic) Data*.

4.3.5. Operations-Related Assets (AM5)

Operation-Related Assets are all the assets created as the result of operational activities, extracted during the operational activities, or used during the operational activities, e.g., any data collected during product use (see Figure 10). The observed operations-related assets include *Customer Data*, *Application Data*, and *Usage Data*. Table 9 presents operations-related assets, their properties, and definitions.

4.3.6. Environment-and-Infrastructure-Related Assets (AM6)

Environment-and-Infrastructure-Related Assets are all the assets used in the development environment or as an infrastructure for development during software development (see Figure 11). The observed environment-and-infrastructure-related assets include *Deployment Infrastructure*, *Tools*, and *Tools Pipelines*. Table 10 presents environment-and- infrastructure -related assets, their properties, and definitions.

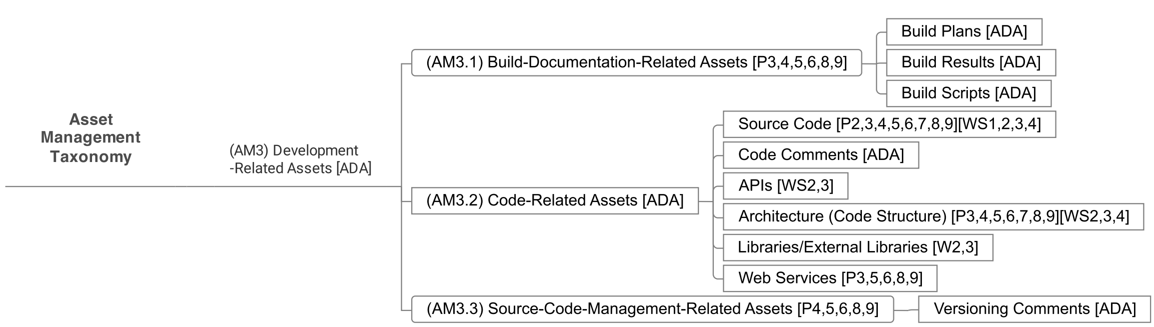

Figure 8: Development-Related Assets Subtree.

Table 7
Development-Related Assets' Definitions.

Asset	AM type	Definition
Build Plans	AM3.1	Build Plans are the descriptions of how developers intend to build the software, i.e., by compilation of artefacts in a build chain, which will end in a running software.
Build Results	AM3.1	Build Results are the results of the build process, including the comments, documentation, and other artefacts that are generated during the build process.This is seen as a persistent asset if it holds more data than just an automated "throw away" report, and/or if the asset is used for reference over time.
Build Scripts	AM3.1	Build Scripts are the scripts that are used to run the build process.
Source Code	AM3.2	Source Code is the collection of code written in a human-readable and comprehensible manner stored as plain text [31].
Code Comments	AM3.2	Code Comments are the comments that developers integrate and write within the source code to clarify and describe certain parts of the code or its functionality [27].
APIs	AM3.2	APIs (Application Program Interfaces) are the interfaces that are created to facilitate interaction of different components and modules.
Architecture (Code Structure)	AM3.2	Architecture is the actual and fundamental relationships and structure of a software system and its source code [5].
Libraries/ External Libraries	AM3.2	Libraries/External Libraries are source code that belongs to the product but is not developed or maintained within the project, i.e., the developers. The software project depends on it and references the library.
Web Services	AM3.2	Web Services are running services on devices handling requests coming from networks
Versioning Comments	AM3.3	Versioning comments are the comments that developers submit to any version control application they use for the development. Such comments can later be extracted and viewed to identify the purpose of each event.

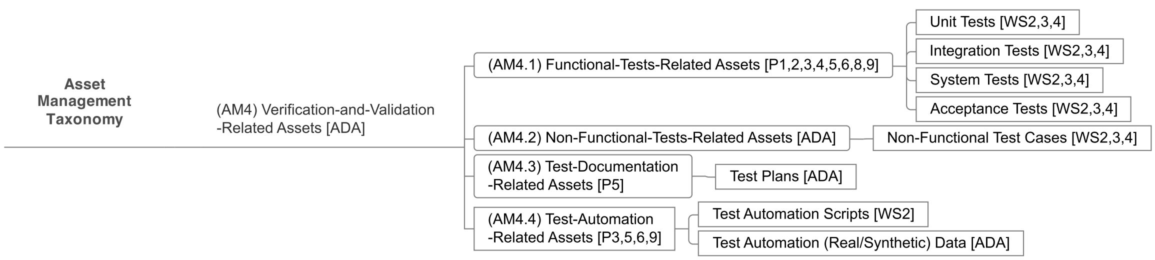

Figure 9: Verification-and-Validation-Related Assets Subtree.

4.3.7. *Development-Process/Ways-of-Working-Related Assets (AM7)*

Development-Process / Ways-of-Working-Related Assets include the following three types of assets (see Figure 12). Table 11 presents development-process / ways-of-working -related assets, their properties, and definitions.

Product-Management-Related Assets (AM7.1) refer to the assets related to dealing with the specific software-intensive product that is being created. These assets come from different stages, such as business justification, planning, development, verification, pricing, and product launch. We have observed the following assets belonging to this type: *Product Management Documentation*, *Documentation About Release Procedure*, *Product Business Models*, *Product Roadmap*, *Product Scope*, and *Product Backlog*.

Process-Management-Related Assets (AM7.2) refer to the assets related to managing the development process, including internal rules, plans, descriptions, specifications, strategies, and standards. We have observed the following assets belonging to this type: *Requirements Internal Standards*, *Ar-*

Table 8
Verification-and-Validation-Related Assets Definitions.

Asset	AM type	Definition
Unit Tests	AM4.1	Unit Tests are the tests written to examine the individual units of the code [28]. "Unit tests generally focus on the program logic within a software component and on correct implementation of the component interface." [7]
Integration Tests	AM4.1	Integration Tests are the tests written to examine the combined set of modules as a group [7, 29].
System Tests	AM4.1	System Tests are the tests written to examine the system's compliance with the requirements.
Acceptance Tests	AM4.1	Acceptance Tests are the tests conducted to examine and determine whether the requirements are met according to the specifications of the requirements.
Non-Functional Test Cases	AM4.2	Non-Functional Test Cases are the tests that examine the quality of the system, i.e., non-functional aspects such as performance, availability, and scalability.
Test Plans	AM4.3	Test Plans are the documents that describe the testing scope and test activities that will be performed on the system throughout the development lifecycle.
Test Automation Scripts	AM4.4	Test Automation Scripts are the scripts that automate part of the testing process. More specifically, the scripts automate distinct testing activities or types of tests.
Test Automation (Real/ Synthetic) Data	AM4.4	Test Automation (Real/Synthetic) Data is the generated data that are used by the automation scripts to test the system.

Figure 10: Operation-Related Assets Subtree.

Table 9
Operations-Related Assets' Definitions.

Asset	AM type	Definition
Customer Data	AM5	Customer Data is data that is collected from the customers (end users) of the software product such as user feedback.
Application Data	AM5	Application Data is the data that is created, collected, used, and maintained while developing the software product such as system performance.
Usage Data	AM5	Usage Data is the data that is collected while the software product is operational such as the data related to the performance of the system.

chitectural Internal Standards, *Documentation Internal Rules / Specifications*, *Build Internal Standards*, *Coding Internal Standards / Specifications*, *Versioning Internal Rules / Specifications*, *Testing Internal Rules / Specifications / Plans / Strategies*, *Process Internal Descriptions*, *Process Data*, and *Documentation About Ways of Working*.

4.3.8. *Organisation-Related Assets (AM8)*

Organisation-Related Assets are all the assets that represent organisations' properties (see Figure 13). The observed organisation-related assets include Organisation's Structure, Organisation's Strategy, Human Capital, and Business Models. Table 12 presents organisation-related assets, their properties, and definitions.

5. Discussion

In this section, we first discuss our findings in light of the research questions, followed by the general lessons learned and implications.

5.1. Principal Findings

RQ : *What assets are important for organisations during the inception, planning, development, evolution, and maintenance of SIPS?*

We present a taxonomy of assets, which includes eight major types of assets *AM1* to *AM8*. The types of assets belonging to each of the major types of assets are not isolated, i.e., some types of assets and assets in them are interrelated. For example, the asset *architectural documentation* is directly related to the asset *architecture*. Architectural documentation represents the architecture of the system.

Out of the eight major types of assets, two types have

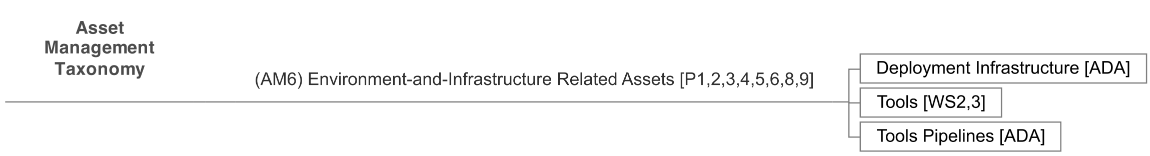

Figure 11: Environment-and-Infrastructure-Related Assets Subtree.

Table 10
Environment-and-Infrastructure-Related Assets' Definitions.

Asset	AM type	Definition
Deployment Infrastructure	AM6	Deployment Infrastructure are all the steps, activities, tools, process descriptions, and processes that facilitate the deployment of a software-intensive product.
Tools	AM6	Tools are any physical and virtual entities that are used for the development of a software product such as integrated development environments (IDE), version control systems, spreadsheets applications, compilers, and debuggers.
Tools Pipelines	AM6	Tool Pipelines are automated processes and activities that facilitate and enable developers to reliably and efficiently compile, build, and deploy the software-intensive product.

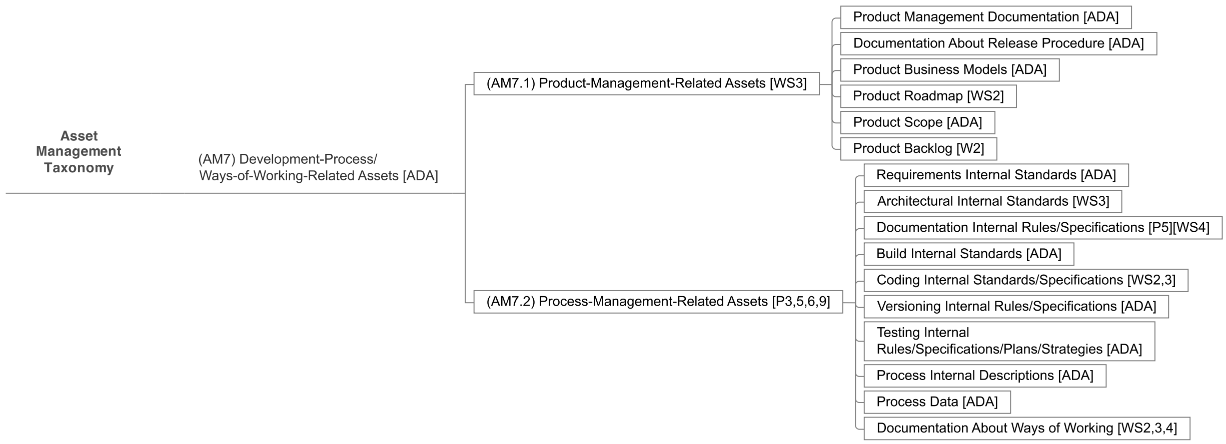

Figure 12: Development-Process/Ways-of-Working-Related Assets Subtree.

Table 11
Development-Process/Ways-of-Working-Related Assets' Definitions.

Asset	AM type	Definition
Product Management Documentation	AM7.1	Product Management Documentation is any documentation that is used to facilitate the management activities and processes during the product development.
Documentation About Release Procedure	AM7.1	Documentation About Release Procedure is the description of the product release plan and the entities and activities associated with release.
Product Business Models	AM7.1	Product Business Models are the descriptions of how the organisation creates value for the customers with the software-intensive product.
Product Roadmap	AM7.1	Product Road Map is the abstract, high-level description of the evolution of the product during the development.
Product Scope	AM7.1	Product Scope is the description of the characteristics, functionality, and features of the software-intensive product.
Product Backlog	AM7.1	Product Backlog is any document that acts as a list where the features, change requests, bug fixes, and other similar activities are stored, listed, and prioritised.
Requirements Internal Standards	AM7.2	Requirements Internal Standards are the specific rules that the company introduces and utilises internally for dealing with the requirements of the product.
Architectural Internal Standards	AM7.2	Architectural Internal Standards are the specific rules that the development team introduces and utilises internally for designing, creating, and maintaining the architecture of the software-intensive product.
Documentation Internal Rules / Specifications	AM7.2	Documentation Internal Rules/Specifications are the specific rules that the development team introduces and utilises internally for creating and maintaining the documentation.
Build Internal Standards	AM7.2	Build Internal Standards are the specific rules that the development team introduces and utilises internally for the build activities.
Coding Internal Standards/Specifications	AM7.2	Coding Internal Standards/Specifications are the rules that the development team introduces and utilises internally while developing the software-intensive product.
Versioning Internal Rules / Specifications	AM7.2	Versioning Internal Rules/Specifications are the rules that the development team introduces and utilises internally for version control during the development of software-intensive products.
Testing Internal Rules / Specifications / Plans / Strategies	AM7.2	Testing Internal Rules/Specifications/Plans/Strategies are the rules that the development team introduces and utilises internally for testing activities and procedures.
Process Internal Descriptions	AM7.2	Process Internal Descriptions are the descriptions of the procedures and activities that the development team introduce and utilise during the development of software-intensive products.
Process Data	AM7.2	Process Data is are the metrics and other information that concern the past and current status of the development process. Examples of such data are velocity, issues, bugs, backlog items, etc.
Documentation About Ways of Working	AM7.2	Documentation About Ways of Working are the description of work plans and working patterns, i.e., how the organisation and the development team plan to create and release the software-intensive product.

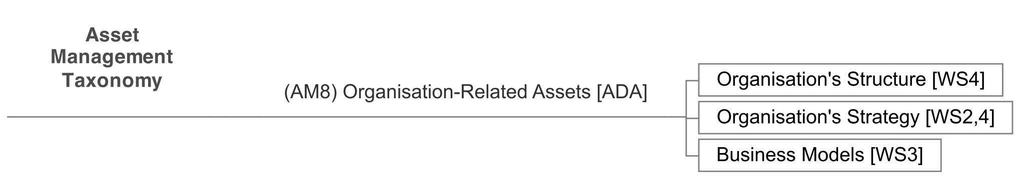

Figure 13: Organisation-Related Assets Subtree.

Table 12
Organisation-Related Assets' Definitions.

Asset	AM type	Definition
Organisation's Structure	AM8	Organisation's Structure is the description of how the organisation directs the activities to achieve organisational goals.
Organisation's Strategy	AM8	Organisation's Strategy is the description of the plans that guide the organisation how to allocate its resources to support the development of the software-intensive product.
Business Models	AM8	Business Models are the descriptions of how the organisation creates value with the software-intensive product for the organisation.

been studied more extensively, namely *Development-Related* Assets and *Product-Representation-Related* Assets, which are aligned with previous studies such [4], i.e., prior studies focus on source-code-related assets. These types of assets are easier to study due to the abundance of metrics and evaluation methods and, therefore, have been studied in many articles. The reason behind this might be that:

- The technical debt metaphor was initially introduced in the context of these prevalent assets [4]. Therefore the researchers had more time investigating and exploring this specific phenomenon. For example, many papers investigate a software product's architecture, exploring different ways of evaluating architecture using different tools and measurements.

- These types of assets are easier to contextualise in the TD metaphor, i.e., identifying such assets and how they can be subject to incur debt. For example, the concept of code smells is easier to grasp since it is a more tangible artefact. It is simple to define how the software product can incur debt if the code is not "quite right"; i.e., it is smelly.

The rest of the types of assets have not received extensive time to be explored. The reason behind this might be that:

- These types of assets were added later as "types of technical debt," such as Requirements Debt and Process Debt. The technical debt metaphor was not initially used to deal with these types of artefacts [4]. These types of TD were introduced in an effort to extend the metaphor and, therefore, have not been investigated thoroughly.

- Unlike the other types (i.e., *Development-Related* and *Product-Representation-Related* Assets), it is harder to identify and/or define how and to what extent they can incur debt in software products. For example, incurring Documentation Debt might differ in different companies and development teams.

- These types of assets are intertwined with the context of the development process, the culture of the company and the development team, and their standards and way-of-work. What is considered debt might be different depending on the context.

We have seen that the existing literature on TD classifies various TD types and presents ontologies on the topic. These classifications have evolved since the introduction of the extended TD metaphor. We observe that the relevant asset categories we have extracted from industrial insights can be mapped to the classifications provided in TD literature. We observe that:

- Some existing TD types and categories, such as code that are well-defined and well-recognised fit into similar categories as in the presented taxonomy. The industrial insights and definitions of such assets are consistent with the definitions from TD literature.

- Some asset types that are relevant to the industry have been understudied. There is room for extending the research in such areas, e.g., *Operations-Related Assets (AM5)* and *Environment-and-Infrastructure-Related Assets (AM6)*, which is noticeable when examining Table 3, i.e., more assets and types of assets are in the top rows of the matrix (see Section 4.1).

- By creating the taxonomy, we highlight both the areas of interest and the gaps in research. Therefore, identifying the areas in the software engineering body of knowledge that need to be investigated and the areas that need to evolve according to the current interest.

5.2. Lessons Learned

In this section, we will present the lessons learned from running the industrial workshops, synthesising the findings, and creating the taxonomy. The importance of source-code-related assets is undeniable (i.e., assets in *AM1*, *AM2*, *AM3*, and *AM4*). However, we observe that the social and organisation aspect of the development is very important to industry though these aspects have not received as much attention in the TD area [4, 55]. Taking a look at some statements from participants in the industrial workshops highlights this fact. Examples of such statements are:

- *"There are many people who work in the same area in the same code base. Creates conflicts and slow releases."*

- *"The problem is the delta operation, and the plan is at such a high level that it is impossible to understand. Too abstract."*

- *"... training the teams in what is considered best practices improves team cohesion and eases collaboration."*

- *"[There is] no holistic platform strategy (Conway's law)."*

The large-scale software projects developed in large organisations are highly coupled with the social and organisation aspect of work. The prevalence of assets related to the social and organisation aspect of development, e.g., *Business Models* and *Product Management Documentation*, indicates the necessity to characterise and standardise such assets, how they are perceived, and how they are measured and monitored.

While creating the taxonomy, we observed that assets do not exist in isolation, i.e., they are entities with characteristics and properties that exist in a software development ecosystem. In the following, we will discuss the assets with similar characteristics and properties and assets that have implicit relations between each other.

Assets that have similar characteristics and properties. For example, *Unit Tests* have similar characteristics and properties as *Source Code*, i.e., unit tests are code, and therefore, their value degradation can have analogous connotations. This means that there are possibilities to evaluate such assets' degradation with similar characteristics and properties using similar metrics. Still, the degradation of one (e.g., Unit Tests) regarding the other (e.g., Source Code) is also relevant.- Moreover, the degradation of one asset (e.g., Source Code) might impact the degradation of the other asset (e.g., Unit Tests). Therefore, such coupling and relations of the assets should be considered when analysing and managing such assets.

Assets that have implicit relations between each other. Implicit relations between assets can arise from their inherent coupling properties. Different assets related to certain aspects of the product will have implicit relations that are not visible in the taxonomy as presented now. For example, *Architectural Models* and *Architecture (Code Structure)* have an inherent relationship. Architectural Models should be the representation of the architecture of the system, i.e., the code structure. Therefore, similar to the previous point, the value degradation of the assets with such implicit relations can have analogous connotations. Their degradation might impact the degradation of the other assets in the relationship. For example, the degradation of any of the functional-requirements-related assets will eventually be reflected in the degradation of functional-test-related assets.

5.3. Contributions

The contribution of this work is the following:

- Providing common terminology and taxonomy for assets that are utilised during software development and;
- Providing a mapping over the assets and the input used to create the taxonomy, i.e., input from the literature and input from the industry

One contribution of the taxonomy is that it is a guideline for future research by providing a map of different types of assets. The map illustrates the different areas that are defined and studied, and the ones that are not standardised or underexplored. Therefore, the taxonomy provides a summary of the body of knowledge by linking empirical studies with industrial insights gathered through the industrial workshops. Providing a common taxonomy and vocabulary:

- Makes it easier for the communities to communicate the knowledge.
- Creates the opportunity to find and build upon previous work.
- Helps to identify the gaps by linking the empirical studies to the taxonomy.
- Highlights the areas of interest.
- Makes it possible to build and add to the taxonomy (new assets, details) as knowledge is changed over time by researchers in the field.
- For practitioners, the evolving taxonomy can be used as a map of different assets that are normally not associated with the implications of degradation, and potentially the implications of said degradation can be traced.

Another minor contribution is that the taxonomy helps draw out the assets with similar characteristics and implicit relations between each other. Most of such similarities of characteristics, properties, and relations are not immediately visible when considering the assets from certain perspectives. Taking a more abstract and high-level look at the assets involved in the development of software-intensive products can help facilitate the management activities and the overall development process.

Finally, large companies deal with developing SIPS, and they rely on external resources to help them achieve the business goals of their products. A major external contributor to new knowledge that can help the practitioners in the industry is research findings. Therefore, understanding and applying the research findings is crucial for them. Having a taxonomy of assets summarising the state-of-the-art and state-of-practice body of knowledge for the assets utilised for developing software-intensive products is useful. Practitioners can refer to the taxonomy systematically built with the accumulated knowledge of academia and other practitioners to extract what they need in specific domains.

We believe that our observations and effort to bring the different assets and terminologies used to describe the assets will help practitioners be more aware of each type of assets and how they are managed in the context.

5.4. Limitations and Threats to Validity

Similar to any other research work, this research has its limitations and threats to validity. In this section, we cover the limitations of our work and how they might affect the results. The asset management taxonomy dimensions, both asset types and assets, do not represent an exhaustive list. The taxonomy is created based on the data extracted from the literature review and the industrial workshops. We combined the inputs from literature and industry to create the taxonomy. We designed the taxonomy to be extendable with new data identified by us and others in future studies as software engineering areas evolve. We encourage researchers and practitioners to consider the taxonomy within their organisation and identify the potential new asset types and assets that can complement the asset management taxonomy's representativeness.

In the rest of this section, we cover the threats to construct, internal, and external validity as suggested by Runeson and Höst [56] and Runeson et al. [57].

- *Construct validity* reflects the operational measurements and how the study represents what is being investigated. Our research uses two primary sources of data, namely the literature review and the industrial workshops. We are aware that the literature review is not inclusive, and it is within a limited area, i.e., Technical Debt. We chose the technical debt topic for the reasons mentioned in Section 3.2. We acknowledge that limiting the literature review to a specific topic might affect the construct validity of this work, i.e., the input from literature. We are also aware that the participants' statements in the workshops can be interpreted differently by the researcher and the participants. We mitigate this threat in two ways. First, by having two researchers coding the raw data independently; and second, by choosing to code the data using the *in vivo* coding method, the qualitative analysis prioritises the participants' opinions.

- *External validity* refers to the generalisability of the results and whether the results of a particular study can hold in other cases. We acknowledge and understand that the results are not comprehensive and might not be generalisable. The created taxonomy is based on the collected data and is extendable. Additionally, the taxonomy was designed to be agnostic to processes, but since some of the assets are more associated with certain processes, the taxonomy cannot be truly agnostic. Finally, other threats that can affect the study's external validity are the number of involved companies, the country where the companies (investigated sites) are located, i.e., Sweden, and involvement of all the roles in these organisations.

- *Reliability* refers to the extend that the data and analysis is dependent of the researchers. When conducting qualitative studies, the threat to validity is the replicability of the results and the process [46]. In the case of our study, the context and the participants of the workshops are unique and therefore not repeatable. But there is an acceptable margin of validity on the results when conducting qualitative research [46]. We have tried to mitigate this threat by relying on consistency in both when conducting the workshops and the analysis.

6. Conclusions and Future Work

This paper presents a taxonomy for classifying assets that have inherent value for an organisation and are subject to degradation. These assets are used during the development of SIPS. The taxonomy is created and built upon the data extracted from a literature review and industrial workshops. The authors completed the taxonomy by identifying the assets that were not mentioned during the workshops or the literature review, i.e., author defined assets (ADA). This work, i.e., the creation of the taxonomy of assets, attempts to provide an overarching perspective on various assets for academicians and practitioners. The taxonomy allows us to characterise and organise the body of knowledge by providing a common vocabulary of and for assets.

We have addressed the research question by creating the taxonomy and defining the types of assets and the assets that belong to those types. Eight major asset types are introduced in the taxonomy: assets related to *Product Requirements*, *Product Representation*, *Development*, *Verification and Validation*, *Operations*, *Environment and Infrastructure*, *Development Process/ Ways-of-Working*, and *Organisation*. The taxonomy could be used for:

- Identify the gaps in research by providing the points of interest from practitioners' perspectives.

- Identify the state-of-the-art research for individual assets and their properties for practitioners.

- Communicate and disperse the body of knowledge.

The dimensions provided by our taxonomy are not exhaustive. Therefore, we intend to conduct further investigation to complement the taxonomy by incorporating the new knowledge. Furthermore, we would like to study assets that co-occur in management activities and how they impact each other and the development process. Lastly, we intend to investigate the individual properties of assets to identify the metrics used for measuring the asset (or lack thereof). The metrics will be investigated to evaluate how they can help us present the state of assets and their degradation.

Besides, future and ongoing work will use the taxonomy as a base for further studies and exploration of assets, their characteristics, and the concepts of value, degradation, and different types of degradation. Finally, we would like to investigate how the taxonomy can be utilised practically in the industry beyond the mainstream of (mostly) code related tools and methods.

A. Appendix: The Asset Management Taxonomy

The full tree of the asset management taxonomy is presented in Figure 14.

References

[1] Alves, N.S., Mendes, T.S., de Mendonça, M.G., Spínola, R.O., Shull, F., Seaman, C., 2016. Identification and management of technical

debt: A systematic mapping study. Information and Software Technology 70, 100–121.
[2] Alves, N.S., Ribeiro, L.F., Caires, V., Mendes, T.S., Spínola, R.O., 2014. Towards an ontology of terms on technical debt, in: 2014 Sixth International Workshop on Managing Technical Debt, IEEE. pp. 1–7.
[3] Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Avgeriou, P., Stamelos, I., 2018. Reusability index: A measure for assessing software assets reusability, in: International Conference on Software Reuse, Springer. pp. 43–58.
[4] Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C., 2016. Managing technical debt in software engineering (dagstuhl seminar 16162), in: Dagstuhl Reports, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[5] Bass, L., Clements, P., Kazman, R., 2003. Software architecture in practice. Addison-Wesley Professional.
[6] BenIdris, M., 2020. Investigate, identify and estimate the technical debt: a systematic mapping study. Available at SSRN 3606172 .
[7] Berner, S., Weber, R., Keller, R.K., 2005. Observations and lessons learned from automated testing, in: Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005., pp. 571–579. doi:10.1109/ICSE.2005.1553603.
[8] Besker, T., Martini, A., Bosch, J., 2017. Time to pay up: Technical debt from a software quality perspective., in: CIbSE, pp. 235–248.
[9] Besker, T., Martini, A., Bosch, J., 2018. Managing architectural technical debt: A unified model and systematic literature review. Journal of Systems and Software 135, 1–16.
[10] Blum, B.I., 1994. A taxonomy of software development methods. Communications of the ACM 37, 82–94.
[11] Bourque, P., Fairley, R.E., et al., 2014. Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0. IEEE Computer Society Press.
[12] Broughton, V., 2015. Essential classification. Facet Publishing.
[13] Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., et al., 2010. Managing technical debt in software-reliant systems, in: Proceedings of the FSE/SDP workshop on Future of software engineering research, pp. 47–52.
[14] Broy, M., 2018. A logical approach to systems engineering artifacts: semantic relationships and dependencies beyond traceability—from requirements to functional and architectural views. Software & Systems Modeling 17, 365–393.
[15] Cicchetti, A., Borg, M., Sentilles, S., Wnuk, K., Carlson, J., Papatheocharous, E., 2016. Towards software assets origin selection supported by a knowledge repository, in: 2016 1st International Workshop on Decision Making in Software ARCHitecture (MARCH), IEEE. pp. 22–29.
[16] Coghlan, D., Brannick, T., 2014. Doing Action Research in Your Own Organization (4th ed.). London: Sage.
[17] Constantopoulos, P., Doerr, M., 1995. Component classification in the software information base. Object-Oriented Software Composition , 177.
[18] Cunningham, W., 1992. The wycash portfolio management system. ACM SIGPLAN OOPS Messenger 4, 29–30.
[19] Falessi, D., Shaw, M.A., Shull, F., Mullen, K., Keymind, M.S., 2013. Practical considerations, challenges, and requirements of tool-support for managing technical debt, in: 2013 4th International Workshop on Managing Technical Debt (MTD), IEEE. pp. 16–19.
[20] Fernández-Sánchez, C., Díaz, J., Pérez, J., Garbajosa, J., 2014. Guiding flexibility investment in agile architecting, in: 2014 47th Hawaii International Conference on System Sciences, IEEE. pp. 4807–4816.
[21] Fernández-Sánchez, C., Garbajosa, J., Yagüe, A., Perez, J., 2017. Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study. Journal of Systems and Software 124, 22–38.
[22] Fox, M., Green, G., Martin, P., 2007. Doing practitioner research. Sage.
[23] Garriga, M., 2017. Towards a taxonomy of microservices architectures, in: International Conference on Software Engineering and Formal Methods, Springer. pp. 203–218.
[24] Glass, R.L., Vessey, I., 1995. Contemporary application-domain taxonomies. IEEE Software 12, 63–76.
[25] Glass, R.L., Vessey, I., Ramesh, V., 2002. Research in software engineering: an analysis of the literature. Information and Software technology 44, 491–506.
[26] Griffith, I., Taffahi, H., Izurieta, C., Claudio, D., 2014. A simulation study of practical methods for technical debt management in agile software development, in: Proceedings of the Winter Simulation Conference 2014, IEEE. pp. 1014–1025.
[27] Grubb, P., Takang, A.A., 2003. Software maintenance: concepts and practice. World Scientific.
[28] Hamill, P., 2004. Unit test frameworks: tools for high-quality software development. " O'Reilly Media, Inc.".
[29] ISO/IEC/IEEE, 2010. Systems and software engineering. ISO/IEC/IEEE 24765:2010. Technical Report. ISO/IEC/IEEE.
[30] ISO/IEC/IEEE, 2014. Asset management - Overview, principles, andsch terminology ISO/IEC/IEEE 55000:2014. Technical Report. ISO/IEC/IEEE.
[31] Kernighan, B.W., 1974. 'programming in c- a tutorial. Unpublished internal memorandum, Bell Laboratories .
[32] Kroll, P., Kruchten, P., 2003. The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP: A Practitioner's Guide to the RUP. Addison-Wesley Professional.
[33] Kruchten, P., 2000. The rational unified process 2nd edition: An introduction.
[34] Kruchten, P., 2004. The rational unified process: an introduction. Addison-Wesley Professional.
[35] Kruchten, P., Nord, R., Ozkaya, I., 2019. Managing Technical Debt: Reducing Friction in Software Development. Addison-Wesley Professional.
[36] Kruchten, P., Nord, R.L., Ozkaya, I., 2012. Technical debt: From metaphor to theory and practice. Ieee software 29, 18–21.
[37] Kruchten, P.B., 1995. The 4+ 1 view model of architecture. IEEE software 12, 42–50.
[38] Kujala, S., Miron-Shatz, T., 2013. Emotions, experiences and usability in real-life mobile phone use, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1061–1070.
[39] Kumar, A., Nori, K.V., Natarajan, S., Lokku, D.S., 2014. Value matrix: From value to quality and architecture, in: Economics-Driven Software Architecture. Elsevier, pp. 205–240.
[40] Kwasnik, B.H., 1992. The role of classification structures in reflecting and building theory. Advances in Classification Research Online 3, 63–82.
[41] Law, E.L.C., Van Schaik, P., 2010. Modelling user experience–an agenda for research and practice. Interacting with computers 22, 313–322.
[42] Lehman, M., 1996. Laws of software evolution revisited, in: European Workshop on Software Process Technology, Springer, Berlin, Heidelberg. pp. 108–124.
[43] Lehman, M.M., 1979. On understanding laws, evolution, and conservation in the large-program life cycle. Journal of Systems and Software 1, 213–221.
[44] Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., Fontana, F.A., 2019. Technical debt prioritization: State of the art. a systematic literature review. arXiv preprint arXiv:1904.12538 .
[45] Letouzey, J.L., Ilkiewicz, M., 2012. Managing technical debt with the sqale method. IEEE software 29, 44–51.
[46] Leung, L., 2015. Validity, reliability, and generalizability in qualitative research. Journal of family medicine and primary care 4, 324.
[47] Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. Journal of Systems and Software 101, 193–220.
[48] von Linné, C., 1735. Systema naturae; sive, Regna tria naturae: systematice proposita per classes, ordines, genera & species. Haak.
[49] Méndez, D., Böhm, W., Vogelsang, A., Mund, J., Broy, M., Kuhrmann, M., Weyer, T., 2019. Artefacts in software engineering: a fundamental positioning. Software & Systems Modeling 18, 2777–

2786.
[50] Méndez, D., Penzenstadler, B., Kuhrmann, M., Broy, M., 2010. A meta model for artefact-orientation: fundamentals and lessons learned in requirements engineering, in: International Conference on Model Driven Engineering Languages and Systems, Springer. pp. 183–197.
[51] Miles, M.B., Huberman, A.M., Saldaña, J., 2014. Qualitative data analysis: A methods sourcebook. 3rd.
[52] Müter, L., Deoskar, T., Mathijssen, M., Brinkkemper, S., Dalpiaz, F., 2019. Refinement of user stories into backlog items: Linguistic structure and action verbs, in: International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer. pp. 109–116.
[53] Northrop, L., Clements, P., Bachmann, F., Bergey, J., Chastek, G., Cohen, S., Donohoe, P., Jones, L., Krut, R., Little, R., et al., 2007. A framework for software product line practice, version 5.0. SEI.–2007–http://www. sei. cmu. edu/productlines/index. html .
[54] Power, K., 2013. Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options, in: 2013 4th International Workshop on Managing Technical Debt (MTD), IEEE. pp. 28–31.
[55] Rios, N., de Mendonça Neto, M.G., Spínola, R.O., 2018. A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. Information and Software Technology 102, 117–145.
[56] Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering 14, 131–164.
[57] Runeson, P., Host, M., Rainer, A., Regnell, B., 2012. Case study research in software engineering: Guidelines and examples. John Wiley & Sons.
[58] Saher, N., Baharom, F., Ghazali, O., 2017. Requirement change taxonomy and categorization in agile software development, in: 2017 6th International Conference on Electrical Engineering and Informatics (ICEEI), IEEE. pp. 1–6.
[59] Saldaña, J., 2015. The coding manual for qualitative researchers. Sage.
[60] Silva, M., Oliveira, T., Bastos, R., 2008. Software artifact metamodel: An approach to software artifact authoring .
[61] Šmite, D., Wohlin, C., Galviņa, Z., Prikladnicki, R., 2014. An empirically based terminology and taxonomy for global software engineering. Empirical Software Engineering 19, 105–153.
[62] Sommerville, I., 2015. Software engineering. 10th, in: Book Software Engineering. 10th, Series Software Engineering. Addison-Wesley.
[63] Stringer, E.T., 2014. Action Research (4th Edition). Thousand Oaks, CA: Sage.
[64] Svahnberg, M., Van Gurp, J., Bosch, J., 2005. A taxonomy of variability realization techniques. Software: Practice and experience 35, 705–754.
[65] Taivalsaari, A., Mikkonen, T., 2018. A taxonomy of iot client architectures. IEEE software 35, 83–88.
[66] Tilley, S., Huang, S., 2002. Documenting software systems with views iii: towards a task-oriented classification of program visualization techniques, in: Proceedings of the 20th annual international conference on Computer documentation, pp. 226–233.
[67] Tom, E., Aurum, A., Vidgen, R., 2012. A consolidated understanding of technical debt .
[68] Tom, E., Aurum, A., Vidgen, R., 2013. An exploration of technical debt. Journal of Systems and Software 86, 1498–1516.
[69] Unterkalmsteiner, M., Feldt, R., Gorschek, T., 2014. A taxonomy for requirements engineering and software test alignment. ACM Transactions on Software Engineering and Methodology (TOSEM) 23, 1–38.
[70] Usman, M., Britto, R., Börstler, J., Mendes, E., 2017. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. Information and Software Technology 85, 43–59.
[71] Vegas, S., Juristo, N., Basili, V.R., 2009. Maturing software engineering knowledge through classifications: A case study on unit testing techniques. IEEE Transactions on Software Engineering 35, 551–565.
[72] Vessey, I., Ramesh, V., Glass, R.L., 2005. A unified classification system for research in the computing disciplines. Information and Software Technology 47, 245–255.
[73] Wohlin, C., 2014a. Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th international conference on evaluation and assessment in software engineering, pp. 1–10.
[74] Wohlin, C., 2014b. Writing for synthesis of evidence in empirical software engineering, in: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–4.
[75] Wohlin, C., Wnuk, K., Smite, D., Franke, U., Badampudi, D., Cicchetti, A., 2016. Supporting strategic decision-making for selection of software assets, in: International conference of software business, Springer. pp. 1–15.
[76] Wolfram, K., Martine, M., Sudnik, P., 2020. Recognising the types of software assets and its impact on asset reuse, in: European Conference on Software Process Improvement, Springer. pp. 162–174.
[77] Zabardast, E., Frattini, J., Gonzalez-Huerta, J., Mendez, D., Gorschek, T., 2021. Asset management in software engineering – what is it after all? arXiv preprint arXiv:2101.07768 .
[78] Zhao, Y., Dong, J., Peng, T., 2009. Ontology classification for semantic-web-based software engineering. IEEE Transactions on Services Computing 2, 303–317.

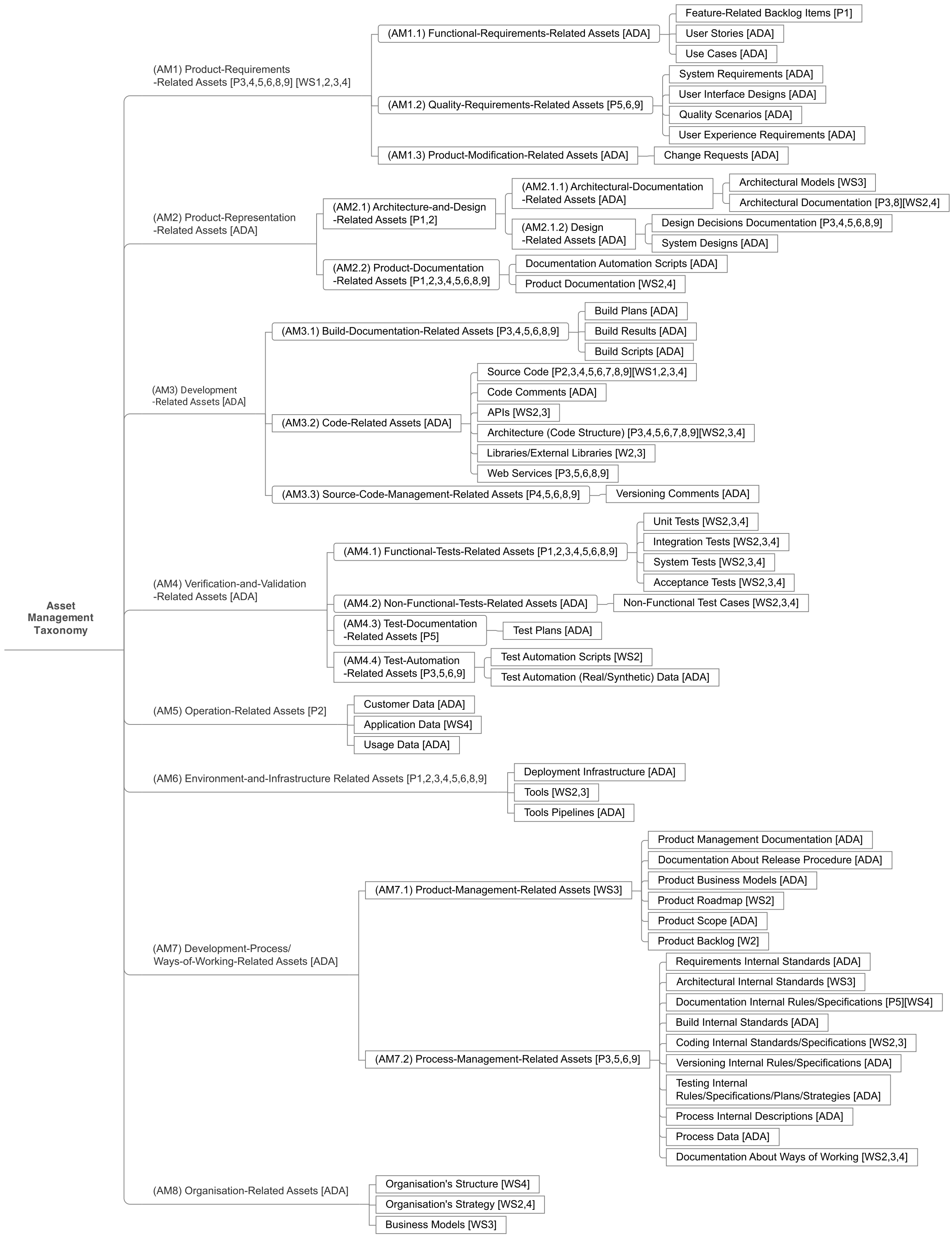

Figure 14: The asset management taxonomy.