The Journal of Systems & Software 202 (2023) 111701

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A taxonomy of assets for the development of software-intensive products and services[☆,☆☆]

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

[a] *Software Engineering Research Lab SERL, Blekinge Institute of Technology, Campus Karlskrona, Valhallavägen 1, Karlskrona, Sweden*
[b] *fortiss GmbH, Guerickestraße 25, 80805 Munich, Germany*
[c] *Department of Computer Science, Aalto University, Espoo, Finland*

ARTICLE INFO

Article history:
Received 21 October 2022
Received in revised form 2 February 2023
Accepted 5 April 2023
Available online 11 April 2023

Keywords:
Assets in software engineering
Asset management in software engineering
Assets for software-intensive products or services
Taxonomy

ABSTRACT

Context: Developing software-intensive products or services usually involves a plethora of software artefacts. Assets are artefacts intended to be used more than once and have value for organisations; examples include test cases, code, requirements, and documentation. During the development process, assets might degrade, affecting the effectiveness and efficiency of the development process. Therefore, assets are an investment that requires continuous management.

Identifying assets is the first step for their effective management. However, there is a lack of awareness of what assets and types of assets are common in software-developing organisations. Most types of assets are understudied, and their state of quality and how they degrade over time have not been well-understood.

Methods: We performed an analysis of secondary literature and a field study at five companies to investigate and identify assets to fill the gap in research. The results were analysed qualitatively and summarised in a taxonomy.

Results: We present the first comprehensive, structured, yet extendable taxonomy of assets, containing 57 types of assets.

Conclusions: The taxonomy serves as a foundation for identifying assets that are relevant for an organisation and enables the study of asset management and asset degradation concepts.

1. Introduction

The fast pace of the development of software-intensive products or services impacts the decision-making process both for design and operational decisions. Such products and services are engineered by "applying well-understood practices in an organised way to evolve a product [or service] containing a nontrivial software component from inception to market, within cost, time, and other constraints Klotins et al. (2018)".

Acting fast to cope with change can compromise the values of the delivered product, environment, development process, and the assets involved, such as source code, test cases, and documentation (Broy, 2018; Zabardast et al., 2022).

Assets are software artefacts that are intended to be used more than once during the inception, development, delivery, or evolution of a software-intensive product or service (Zabardast et al., 2022). Assets can lose their value and degrade "due to intentional or unintentional decisions caused by technical or non-technical manipulation of an asset or its associated assets during all stages of the product life-cycle (Zabardast et al., 2022)".

During the development of software-intensive products or services, decisions are usually made quickly to respond to change, focusing on fast delivery, leaving considerations on the assets involved as a secondary objective. Thus, these decisions often result in long-term negative consequences not only on the quality of the delivered product or service but also on the assets such as code, architecture, and documentation, to name a few.

Researchers and practitioners have traditionally used the technical debt (TD) metaphor (Cunningham, 1992) to refer to the impact of the intentional and unintentional sub-optimal decisions

☆ Editor: Prof W. Eric Wong.

☆☆ This research was supported by the KK foundation through the SHADE KK-Hög project under grant 2017/0176 and Research Profile project SERT under grant 2018/010 at Blekinge Institute of Technology, SERL Sweden.

* Corresponding author.
E-mail addresses: 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@aalto.fi (F. Fagerholm).
URLs: https://ehsanzabardast.com/ (E. Zabardast), http://gonzalez-huerta.net/ (J. Gonzalez-Huerta), https://gorschek.com/ (T. Gorschek).

https://doi.org/10.1016/j.jss.2023.111701
0164-1212/

on assets as a consequence of meeting strict deadlines (Kruchten et al., 2019; Zabardast et al., 2022). In this work, we are investigating in assets' quality degradation; therefore, we explore the TD field as a representative form of quality degradation.

The motivation behind the work presented in this paper comes from industrial collaborations where the concept of TD did not fully cover their needs since code-based assets, although important, only represented a part of the challenge. From a practitioner's perspective, it is valuable to have the ability to identify what assets are available, which of them are important, and how and why they degrade. This is especially important as a product or service evolves over time, when assets are reused many times, and when they inevitably become subject to change and degradation. The degradation of assets is central to their maintenance and evolution (Avgeriou et al., 2016).

In our previous work (Zabardast et al., 2022), we have defined *assets* and *asset degradation*. We have articulated the importance of identifying and studying them. From a research perspective, a potential benefit of introducing *assets* as a complementary concept to TD, with a broader definition, is that it takes all so-called "items of value" (Méndez et al., 2019) into account and widens the view of what can hold value in the context of developing software-intensive products or services (Zabardast et al., 2022). The widened definition also fosters an understanding of what assets can be negatively impacted by degradation and, thereby, the understanding that overlooking said assets can be detrimental to the development and evolution of such products and services. However, to gain such understanding, we must first understand what assets are subject to the concept, warranting the need for the taxonomy of assets presented in this work.

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 analysis of secondary studies and the field study. The results are presented in Section 4, together with the proposed asset management taxonomy. Section 5 discusses 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 software products or services have been studied previously. For instance, from a managerial perspective where the term asset is used to discuss how products in product lines are developed from a set of core assets with built-in variation mechanisms (Northrop et al., 2007), or making emphasis only on business- or market-related assets, like the in the works by Ampatzoglou et al. (2018), Cicchetti et al. (2016), Wohlin et al. (2016) and Wolfram et al. (2020). In contrast, in this work, our focus is on the inception, development, evolution, and maintenance of software-related assets.

2.1. Artefacts and assets in software engineering

Describing how a software system is envisioned, built, and maintained is part of the Software Development Processes (SDP) (Sommerville, 2015). The SDP prescribes the set of activities and roles to manipulate software artefacts, e.g., source code, documentation, reports, and others (Broy, 2018).

Artefacts in software engineering field have been traditionally defined as (i) "documentation of the results of development steps" (Broy, 2018); (ii) "a work product that is produced, modified, or used by a sequence of tasks that have value to a role" (Méndez et al., 2019); 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" (Méndez et al., 2019). Software artefacts are, therefore, self-contained documentation and work products that are produced, modified or used by a sequence of tasks that have value to a role (Méndez et al., 2019).

Understanding software artefacts, how they are structured, and how they relate to each other has a significant influence on how organisations develop software (Broy, 2018). The documentation in large-scale systems can grow exponentially; therefore, there is a need for structurally organising software artefacts (Broy, 2018). Artefacts defined by most of the SDPs are monolithic and unstructured (Tilley and Huang, 2002). The content of poorly structured artefacts is difficult to reuse, and the evolution of such monolithic artefacts is cumbersome (Silva et al., 2009). Therefore, different SDPs present various models for presenting software artefacts, e.g., the Rational Unified Process (RUP) (Kroll and Kruchten, 2003; Kruchten, 2000). There are ways to classify and structure software artefacts based on well-known modelling concepts. Examples of such models are the work of Broy (2018) and Silva et al. (2009). Moreover, there are ontologies and meta-models to classify artefacts in specific software development areas (e.g., Idowu et al. (2022), Méndez et al. (2010), Zhao et al. (2009), and Constantopoulos and Doerr (1995)).

However, the definitions of artefacts presented in the literature do not distinguish between:

i **Artefacts that have an inherent value for the development organisation** (i.e., an asset Zabardast et al., 2022) from the artefacts that do not have any value for the organisation[1] (Zabardast et al., 2022). The value of each asset is a property that can characterise its degradation, i.e., if an asset degrades, although it continues being an asset, its value for the organisation has degraded.

ii **Artefacts that are intended to be used more than once** "An asset is a software artefact that is intended to be used more than once during the inception, development, delivery, or evolution of a software-intensive product or service... Zabardast et al. (2022)" "If an artefact is used once and discarded or disregarded afterwards, it does not qualify as an asset (Zabardast et al., 2022)". We believe that the following example clarifies the distinction between *Temporary Artefacts* vs. *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 test result 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", "tickets" or other management artefacts. Once transformed into the new asset "change requests", it is discarded. New test results reports will be created (and discarded) on each execution of the tests. Therefore, artefacts that are intended to be used more than once and have value for the organisation (i.e., assets) need to be monitored by the organisation since their continuous maintenance and evolution renders the need for exercising quality control (Reussner et al., 2019; Zabardast et al., 2022).

[1] Méndez et al. (2019) define artefacts as having value for a role which is substantially different from having value for the organisation.

2.2. Asset degradation and technical debt

In our previous work (Zabardast et al., 2022), we coined a concept *Asset Degradation* "as the loss of value that an asset suffers due to intentional or unintentional decisions caused by technical or non-technical manipulation of the asset, or associated assets, during all stages of the product life-cycle" (Zabardast et al., 2022). All assets can degrade, which will affect their value for the organisation in different ways. Degradation can be deliberate, unintentional, and entropy (Zabardast et al., 2022). *Deliberate degradation* is introduced by taking a conscious decision, understanding its long-term consequences and accommodating short-term needs. *Unintentional degradation* is introduced by taking a sub-optimal decision either because we are not aware of the other, better alternatives or because we cannot predict the consequences of selecting "our" alternative. Finally, *entropy* is introduced just by the ever-growing size and complexity that occurs when software systems are evolving (Lehman, 1996, 1979). The degradation that is due to the continuous evolution of the software, and which is not coming directly from the manipulation of the asset by the developers, is entropy (Zabardast et al., 2022). We argue that Technical Debt (TD), a metaphor introduced by Cunningham in 1992 (Cunningham, 1992) and which allows reasoning about the compromises resulting from sub-optimal decisions to achieve short-term benefits is one form of quality degradation. All assets (per definition artefacts, too) are subject to TD, i.e., while assets are created, changed, and updated, one might introduce TD on them (Li et al., 2015). We see the introduction of TD aligned with the three types of degradation mentioned above (i.e., deliberate, unintentional, and entropy) as we consider TD to be one form of asset quality degradation.

The TD metaphor has been extended and studied by many researchers (Rios et al., 2018). It has been an interesting topic for both academia and industry, and it has grown from a metaphor to a practice (Kruchten et al., 2012). TD is currently recognised as one of the critical issues in the software development industry (Besker et al., 2017). TD is "pervasive", and it includes all aspects of software development, signifying its importance both in the industry and academia (Kruchten et al., 2019). The activities that are performed to prevent, identify, monitor, measure, prioritise, and repay TD are called Technical Debt Management (TDM) (Avgeriou et al., 2016; Griffith et al., 2014) and include such activities as, for example, identifying TD items in the code, visualising the evolution of TD, evaluating source code state, and calculating TD principal (Rios et al., 2018).

2.3. Taxonomies in software engineering

Scientists and researchers have long used taxonomies as a tool to communicate knowledge. Early examples are noted in the eighteen century, for instance, the work of von Linné (1735). Taxonomies are mainly created and used to communicate knowledge, provide a common vocabulary, and help structure and advance knowledge in a field (Glass and Vessey, 1995; Kwasnik, 1992; Usman et al., 2017). 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 (Broughton, 2015). 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 (Unterkalmsteiner et al., 2014).

Software engineering (SE) is continually evolving and becoming one of the principal fields of study with many sub-areas. Therefore, the researchers of the field are required to create and update the taxonomies and ontologies to help mature, extend, and evolve SE knowledge (Usman et al., 2017). 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 (Bourque et al., 2014). 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. (2002) and Blum (1994). 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 (Taivalsaari and Mikkonen, 2018), Taxonomy of Requirement Change (Saher et al., 2017), Taxonomy of Architecture Microservices (Garriga, 2017), Taxonomy of Global Software Engineering (Šmite et al., 2014), and Taxonomy of Variability Realisation Techniques (Svahnberg et al., 2005) to name a few.

This paper presents a taxonomy of assets in the inception, planning, development, evolution, and maintenance of a software-intensive product or service. 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 3.3.

2.4. Summary of the gaps

In this paper, we identify and categorise assets in software development and software engineering. We use the concept of assets and their intentional and unintentional degradation. Moreover, we aim to address the following real-world problems:

- Bring awareness to practitioners and researchers. A precise and concise terminology of assets enables practitioners and researchers to consider new ways of dealing with asset degradation (Zabardast et al., 2022).
- The ripple effect that the degradation of an asset can impose on other assets is another aspect that necessitates the creation of taxonomy to understand relations between assets (Kruchten et al., 2019; Zabardast et al., 2022).

In this paper, we identify the software artefacts that adhere to this definition of assets and are common in the industry. We aim to address the following gap: identifying and distinguishing assets by considering every aspect of software development. For example, assets related to *environment and infrastructure*, *development Process*, *ways of working*, and *organisational aspects* (Avgeriou et al., 2016; Rios et al., 2018) that have not received enough attention. We aspire to identify assets and provide a synthesis of existing knowledge in the area of asset management.

3. Research overview

This section presents the research methodology of the paper. The process followed to build the taxonomy is divided into two main parts: a systematic analysis of secondary studies and a field study (Stol and Fitzgerald, 2018) of industrial cases using focus group interviews and field notes as data collection methods. Combining an SLR and a field study helps us look at the phenomenon from different perspectives and create a complete picture of the phenomena of assets. In addition, this multi-faceted view provides insights that can strengthen the validity of the results.

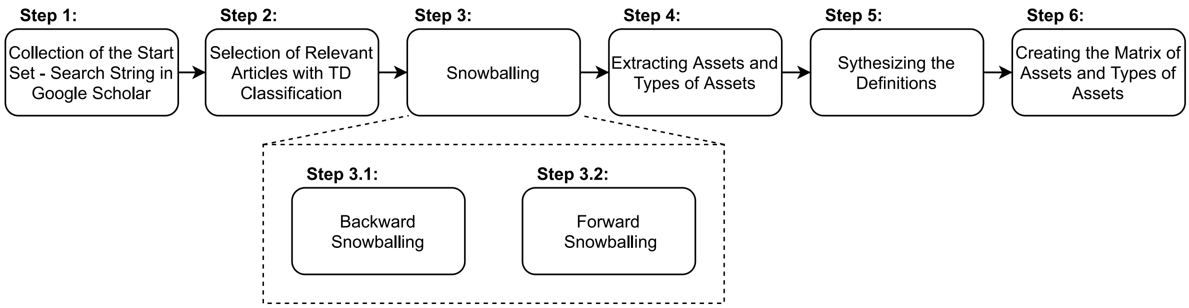

Fig. 1. The execution process of the analysis of the secondary studies.

In this work, we are answering the following research question:

RQ : *What assets are managed by organisations during the inception, planning, development, evolution, and maintenance of software-intensive products or services?*

The rest of this section presents the analysis of secondary studies (See Section 3.1), the field study (Stol and Fitzgerald, 2018) (See Section 3.2), and taxonomy creation (See Section 3.3).

3.1. Analysis of secondary studies: Planning and execution

This subsection describes the systematic analysis of secondary studies conducted in this work. All assets are subject to degradation. Asset degradation can be classified into three categories deliberate, unintentional, and entropy (Zabardast et al., 2022). Asset degradation can also be measured and monitored using different metrics, e.g., the amount of TD. TD is a familiar concept for practitioners and has received a growing interest in academia and the industry (Rios et al., 2018). TD has become a broad research area that has focused on different assets. The fact that TD research studies a particular asset might imply that asset might be of value for software development organisations, or at least that might be perceived as such by TD researchers. Therefore, we do believe we can use the TD literature as a proxy for identifying and categorising different types of assets. Since we are interested in assets' quality degradation, we explore the TD field as a representative form of quality degradation.

In order to study the state of art, we performed an systematic analysis of secondary studies to capture the classifications and definitions of the various assets addressed by these previous research works (top-down method) (Broughton, 2015). In our literature review, the goal was to identify systematic literature reviews, systematic mapping studies, and tertiary studies. We reviewed secondary and tertiary research papers by performing an SLR using snowballing, following the guidelines by Wohlin (2014). We selected snowballing as a search strategy as it allowed us to explore the area as well as its reported efficiency (Badampudi et al., 2015).

The starting set of papers for snowballing was collected through a database search in Google Scholar in October 2021 using the following search string in: *"technical debt" AND ("systematic literature review" OR "systematic mapping study" OR "tertiary study")*. We chose to use the search string only in Google scholar since it is not restricted to specific publishers (Badampudi et al., 2015), and it can help avoid publisher bias (Wohlin, 2014). We selected, with the inclusion and exclusion described below, the articles that presented a classification for TD, i.e., articles that present different types of TD.

The execution procedure to identify assets included the following steps as illustrated in Fig. 1. 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 TD, by using a search string.
- **Step 2:** Evaluate the papers – start set in the first iteration – for inclusion/exclusion based on the criteria.
  - **Inclusion Criteria:**
    - The selected papers should report secondary or tertiary studies;
    - the papers should present any classification of TD and assets affected by TD;
    - the papers should be written in English; the papers' main aim is the literature review.
  - **Exclusion Criteria:**
    - The papers that present informal literature reviews and are duplication of previous studies are not included.
- **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 selected papers. The backward snowballing was finished in one round.
  - **Step 3.2:** Forward snowballing by looking at the papers that cite the selected papers. The forward snowballing was finished in one round.
- **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 TD classifications defined by the selected articles.

3.2. Field study (Focus group interviews): Planning and execution

To study the state of practice, we performed field studies, using focus groups and field notes as data collection in five companies to find evidence on how assets are defined and used. The five companies were selected using convenience sampling, as the companies are involved in an ongoing research project that focuses, among other topics, on addressing asset degradation challenges.

The focus groups' process is presented in Fig. 2. The reports from the focus groups were coded and used for the construction of the taxonomy. We used the bottom-up method (Broughton, 2015) for updating the existing structure that we obtained from the literature review. The focus groups were planned as a half-working day (four hours).

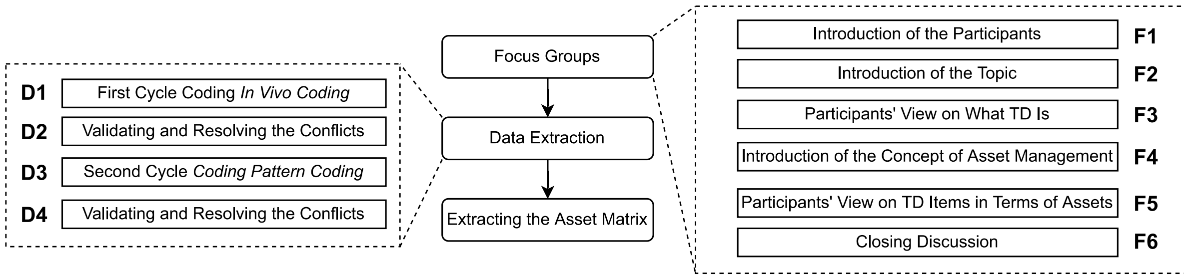

Fig. 2. The field study execution process.

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	Number of participants
Ericsson	Telecommunication & ICT	Karlskrona, Sweden	Large	Senior System Architect Corporate Senior Strategic Expert Operations & Testing	3
Fortnox	Finance	Växjö, Sweden	Large	Head of Development Product Owner Development Manager System Architect Testing	14
Qtema[a]	Consultancy	Stockholm, Sweden	SME	Chairman of the Board Requirements Analyst Sales Manager Project Manager IT Administration Manager	6
Time People Group[a]	Consultancy	Stockholm, Sweden	SME	Data Consultant Project Manager Consultant Senior Agile Coach IT Project Manager Team Leader Chief Executive Officer (CEO) Consultant Test Leader	7
Volvo CE	Construction Machinery	Gothenburg, Sweden	Large	Enterprise Architect Solution Architect Business Information Architect	5

[a]Time People Group and Qtema participated in the same workshop.

3.2.1. Case company characterisation

We have collected the data for this study by collaborating with five companies that work in the area of construction machinery, communication & ICT, consultancy services, and financial services. The research partner companies are Ericsson (telecommunication & ICT), Fortnox (Financial Services), Qtema (Consultancy Services), Time People Group (Consultancy Services), and Volvo CE (Construction Machinery).

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 processes, which turns into their willingness to participate in studies like this. All the collaborating companies work on developing software-intensive products or services and are involved in a large ongoing research partnership.[2] 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 focus groups (focus group IDs) in Table 4, which has been shuffled to preserve confidentiality.

3.2.2. Focus groups' procedure

The steps taken during the industrial focus groups are presented in this section. The focus groups include six steps (see Fig. 2):

- ***F1.*** Focus group participants introducing themselves.
- ***F2.*** One of the moderating researchers presenting the topic.
- ***F3.*** Focus group participants discussing the topic, providing insight into their views/experiences with TD and document them in notes.
- ***F4.*** Focus group participants discussing assets and asset management in detail after a second presentation of the concept.
- ***F5.*** Focus group participants discussing what they wrote before as TD examples and rethinking them in terms of assets, asset degradation, and asset management.
- ***F6.*** A closing discussion and focus groups.

Each Focus group starts with participants introducing themselves with background information about their work, including their current role in the organisation (step F1). One of the moderating researchers then presents the Focus group's agenda and covers the importance of the topic and the growing interest in

2 See www.rethought.se.

value creation and waste reduction both in academia and in the industry (step F2).

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 TD), i.e., the problems they know or have encountered or experienced (step F3). 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 a second presentation, i.e., introducing the participants to the concepts related to asset management and asset degradation (step F4), the participants add new items to the previous notes on the board. Participants then refine the items from the board for the rest of the Focus group (step F5). The Focus group ends with a closing discussion on the topic and the items (step F6).

In the context of this research, we have moved the focus from the traditional TD 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).

The researchers' minutes that were written during each session were then aggregated and summarised in a report sent back to the participants, that were asked to provide us with feedback. The written notes from the participants and the final reports were used as raw data for creating the taxonomy using the data extraction method described in Section 3.2.3. The raw data (i.e., participants' notes and the reports) 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 3.3.

Unfortunately, since this research is under non-disclosure agreements (NDA) with participating companies, we cannot disclose any further information about the companies, the participants, or the collected materials.

3.2.3. Data extraction

To create the matrix of assets from industrial insights, we use the hybrid method of coding, as suggested by Saldaña (2015). 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 Focus group. We have used the *in vivo coding* method (Saldaña, 2015) to label the raw data in the first cycle. In vivo coding prioritises the participants' opinions (Saldaña, 2015); therefore, it is suitable for labelling raw data in the first cycle coding in our study, where participants' opinions are used as input. 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" (Stringer, 2014, p. 140). It is commonly used in empirical and practitioner research (Coghlan and Brannick, 2014; Fox et al., 2007; Stringer, 2014).

The coding was done by two researchers independently (step D1, see Fig. 2). 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 D2, see Fig. 2).

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 (Saldaña, 2015) as the second cycle coding method. Pattern codes are explanatory or inferential codes that identify an emergent theme, configuration or explanation (Saldaña, 2015, p. 237). According to Miles et al. (2014, 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 D3, see Fig. 2). 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 D4, see Fig. 2). The results of the insights gathered from industrial focus groups are presented in Section 4.2.

3.3. 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 Fig. 3). The metamodel illustrates the structural relationships between the metaclasses, i.e., concepts, in the taxonomy. The metaclasses presented in the metamodel are:

- The *"AssetsTaxonomy"* 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. Types of 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 focus groups), or the identified by researchers.
- The *"Asset"* metaclass represents assets. Each asset belongs to one and only one type of asset, assuring orthogonality by design. 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 focus groups), 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 focus groups). References can be mapped to individual *assets/type of assets* or multiple *assets/type of assets*.

The creation of the taxonomy included three steps. First, we summarised the relevant topics on TD 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 focus groups (Section 3.2.3) 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 focus groups.

After the creation of the matrices, all the candidate assets in each matrix were listed and presented to all the researchers that were part of the industrial workshops. The researchers discussed whether each candidate asset and asset type should be included

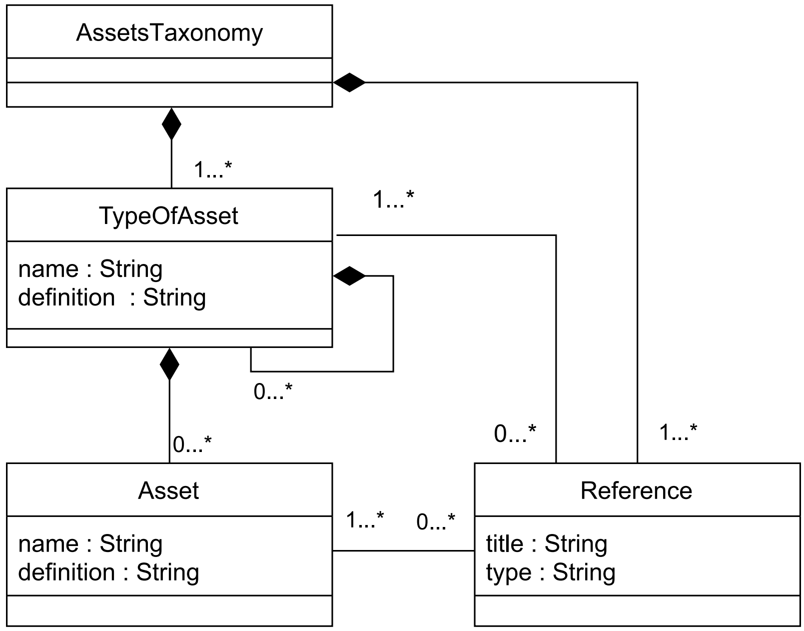

Fig. 3. Asset management metamodel.

in the taxonomy. The discussion included ensuring each candidate asset adhered to two criteria: (1) the candidate asset should adhere to the definition of an asset, and (2) how a candidate asset would degrade. We decided to include a candidate after all the researchers agreed on it adhering to the criterion. The rest of the candidates were discarded.

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 and extensive industry knowledge from decades of industrial research that we perceived 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[3] where all the researchers discussed, reflected, improved, and added or 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. The assets included in the taxonomy should adhere to the definition of an asset (Zabardast et al., 2022).

3.4. Taxonomy validation

This section describes the taxonomy validation procedure. According to Usman et al. (2017), the validation process includes orthogonality demonstration, benchmarking, and utility demonstration. The taxonomy was created to be orthogonal by design (i.e., each element can only be a member of one group), as described in Section 3.3. Therefore, in this validation we focus on benchmarking and utility demonstration.

We conducted three separate workshops (one for each company) to validate the taxonomy and its structure with the six participants (including Product Owner, Developer, Software Architect, Scrum Master, Test Quality Assurance, and Research Engineer) from three companies namely, Ericsson (2 participants), Fortnox (3 participants), and Volvo CE (1 participant) who were involved during the industrial workshops for the data collection. The procedure for the validation workshops is presented below.

1. The taxonomy was sent to the participants to study before the validation workshops.
2. During the validation workshops, the taxonomy was presented to the participants.
3. After the presentation, the participants filled in a questionnaire that included four questions:
   (a) "*Select the assets from the taxonomy that you were not aware of.*"
   (b) "*Select the assets that you think should not be in the taxonomy.*"
   (c) "*Write down the assets that are missing from the taxonomy.*"
   (d) "*Prioritise the top 5 assets based on your experience. (index starting from 1 as the most important asset). You can add missing assets from the previous question as well.*"
4. At the end of each workshop, a discussion session was held where participants asked questions regarding the taxonomy and its content. The discussions were recorded.

We decided to apply changes to the taxonomy after the validation workshops were concluded, in cases when the majority of the participants suggested a given change in the taxonomy. The results of the validation workshops together with the discussion on the results are presented in Section 4.4.

4. Results

This section presents the results of the systematic literature review and then the results from the field study. We will present the asset management taxonomy built by aggregating the results from both. Finally, we will present the results of the validation workshops.

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

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 (Tom et al., 2012)		×	
P2	An Exploration of Technical Debt (Tom et al., 2013)		×	
P3	Towards an Ontology of Terms on Technical Debt (Alves et al., 2014)	×		
P4	A Systematic Mapping Study on Technical Debt and Its Management (Li et al., 2015)	×		
P5	Identification and Management of Technical Debt: A Systematic Mapping Study (Alves et al., 2016)	×		
P6	Managing Architectural Technical Debt: A Unified Model and Systematic Literature Review (Besker et al., 2018)	×		
P7	A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners (Rios et al., 2018)	×		
P8	A systematic literature review on Technical Debt prioritisation: Strategies, processes, factors, and tools (Lenarduzzi et al., 2021)			×
P9	Investigate, identify and estimate the technical debt: a systematic mapping study (BenIdris, 2020)			×

4.1. Results from the analysis of secondary studies

The final list of selected papers included nine articles, presented in Table 2. To create the final assets' matrix (presented in Section 4.3), we extracted the data from each article. For example, in paper *P4* (Li et al., 2015), we refer to Fig. 8 on page ten of the article, where the authors summarise the "TD classification tree" 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" (Li et al., 2015, p.9). Requirements is also mentioned as a TD item in *P3*, *P5*, *P6*, and *P8*. Table 3 presents the summary of our findings based on the types of TD. It is important to mention that the columns P1 to P9 in Table 3 retain the exact extracted words from the data. The "Emerging Category(ies)" and "Phase, during which the artefact is produced" columns are the synthesised information for each row. The codes of the papers are used as a reference throughout this paper. It is important to mention that Table 3 does not represent an overview of the final assets but only the ones extracted from the SLR. However, this matrix helps us to categorise assets and types of assets according to existing classifications.

Looking at Table 3 we can see that there are fewer categories in the earlier studies (*P1* and *P2*) compared to later studies (*P3* to *P9*). The more recent papers that follow these studies break down the bigger categories into more specific categories. For example, *Architecture and Design* are put into one category in *P1* and *P2* and later, they are broken down into 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. Results from the field study

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

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 that discussion.

The focus groups are presented in chronological order in Table 4, i.e., *WS1* was the first focus group. It is important to mention that the columns WS1 to WS4 in Table 4 retain the exact extracted words from the data. The "Emerging Category(ies)" and "Phase, during which the artefact is produced" columns are the synthesised information for each row. Examining Table 4, we observe that assets are mentioned more often than types of assets in the industrial focus groups, 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 focus groups than the literature review.

4.3. The asset management taxonomy

Using the key concepts extracted from the labelled data (presented in Tables 3 and 4), we build the taxonomy of assets. The taxonomy contains the assets identified both through the literature review and through the industrial focus groups. The assets included in the taxonomy are presented in a tree (graph). The nodes represent the assets (the leaf nodes) and the types of assets (non-leaf nodes).

The tree presented in Fig. 4 contains only the types of assets (The full tree is presented in Appendix). Note that the nodes in the tree in Fig. 4 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 focus groups.[4] And finally, Author Defined Assets (*[ADA]*) are assets included in the taxonomy by the researchers.

The process described in Section 3.3, combining the asset matrices from the literature review, the input from the industrial focus groups, and completing the tree with *Author Defined Assets* (ADA) resulted in the taxonomy containing 24 types of assets and a total of 57 assets.

The eight main types of assets included in the taxonomy are:

- **Product-Requirements-Related Assets (AM1)** refer to assets (and types of 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 assets (and types of assets) concerned with system and architectural design and any documentation related to these assets.

4 The IDs on the focus groups on Table 4 have been obfuscated to preserve anonymity and have no relationship with the order of companies shown in Table 1.

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	Testing	Test Defects Test Automation	Test Defects	Test Defects Test Automation Test Case Documentation	Test Defects Test Automation		Test Defects	Test Defects Test Automation	*Functional Test* *Test Automation* *Test Documentation*	Verification and Validation
Infrastructure	Environment Hardware Infrastructure Operational Processes	Infrastructure	Infrastructure	Infrastructure	Infrastructure		Infrastructure	Infrastructure	*Environment & Infrastructure* *Environment & Infrastructure* *Operations*	Operations
		Process		Process Documentation Specifications	Process			Process	*Process Management* *Documentation Internal Rules & Specifications*	Management

Colour Guide: Assets   Types of Assets   Temporary Artefacts.

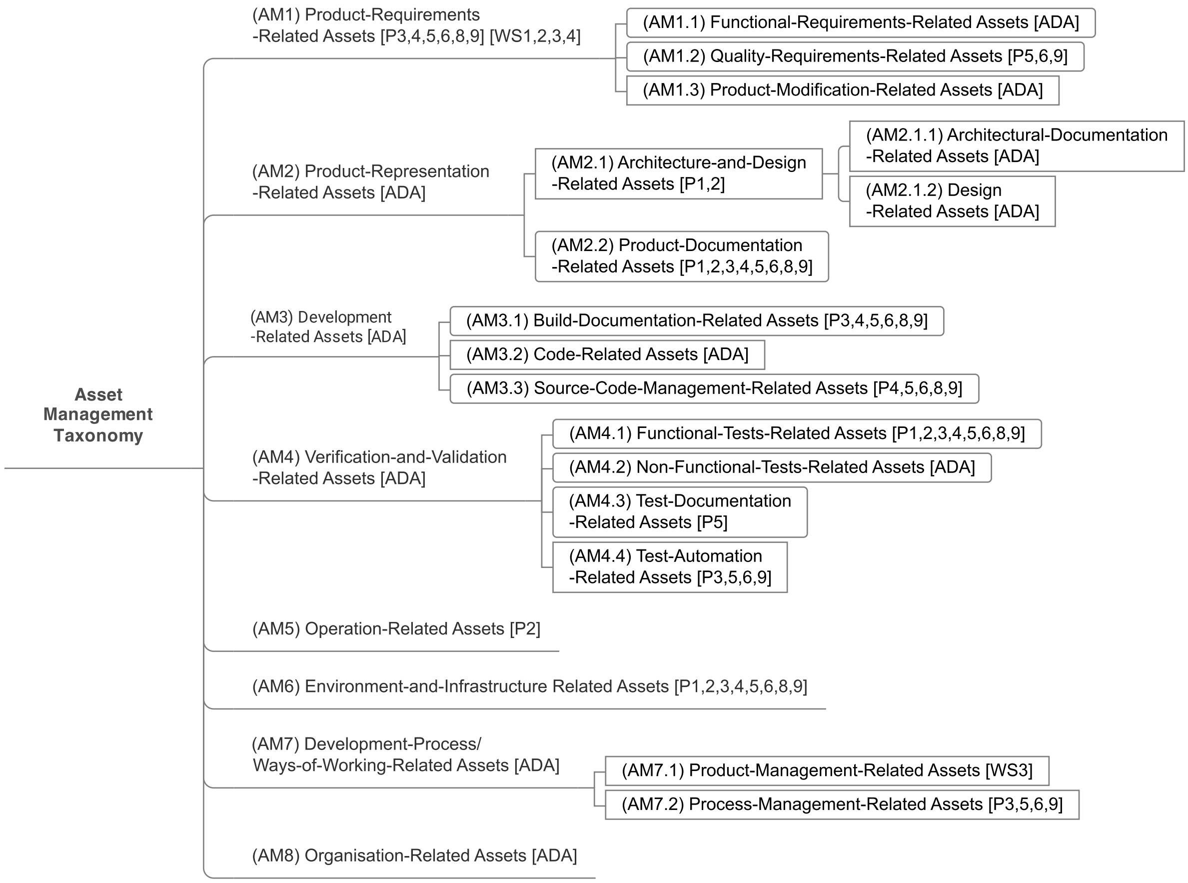

Fig. 4. The asset management taxonomy. The tree contains only the types of assets. The full tree is presented in Appendix.

- **Development-Related Assets (AM3)** refer to the assets (and types of assets) concerned with the development of the software product, including the code, build, and versioning.
- **Verification-and-Validation-Related Assets (AM4)** refer to assets (and types of assets) concerned with software testing and quality assurance and the output provided by such assets that help the stakeholders investigate the quality of the software product.
- **Operations-Related Assets (AM5)** refer to assets (and types of assets) concerned with the data produced or collected from operational activities, e.g., any data collected during the use of the product or service.
- **Environment/Infrastructure-Related Assets (AM6)** refers to assets (and types of assets) concerned with the development environment, the infrastructure, and the tools (including support applications) that facilitate the development or deployment process.
- **Development-Process/Ways-of-Working-Related Assets (AM7)** refer to assets (and types of assets) concerned with product and process management and all the interrelated processes and procedures during the development process.
- **Organisation-Related Assets (AM8)** refer to assets (and types of assets) concerned with organisations, such as team constellation, team collaborations, and organisational governance.

In the remainder of the section, we present the eight major types of assets labelled *AM1-AM8*. We include the definitions of each type of asset together with their corresponding assets. Assets' definitions are presented in Appendix.

4.3.1. Product-Requirements-Related Assets (AM1)

Product-Requirements-Related Assets include the following three types of assets (see Fig. 5):

- *Functional-Requirement-Related Assets (AM1.1)* refer to the assets related to the functions that the software shall provide and that can be tested (Bourque et al., 2014). We have identified 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 (Bourque et al., 2014). We have identified 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 identified belonging to this type.

4.3.2. Product-Representation-Related Assets (AM2)

Product-Representation-Related Assets include the following two types of assets (see Fig. 6):

- *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:

Table 4
Asset matrix from industrial input.

WS1	WS2	WS3	WS4	Emerging category(ies)	Phase, during which the artefact is produced
Contradictory Requirements[a]	Requirements[a]	Requirements[a]	Requirements[a]	*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
	Kubernetes	Containers \Kubernetes	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

Colour Guide: Assets Types of Assets Temporary Artefacts.
The term *requirements* refers to software requirement area. We use requirement artefact when referring to documents such as SRS.

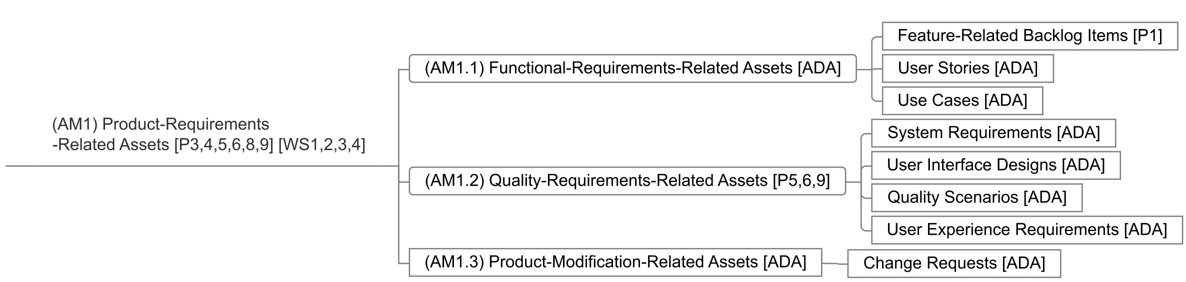

Fig. 5. Product-Requirements-Related Assets Sub-tree.

– *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 identified 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 that occurs during the development process. We have identified 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 identified the following assets belonging to this type: *Documentation Automation Scripts* and *Product Documentation*.

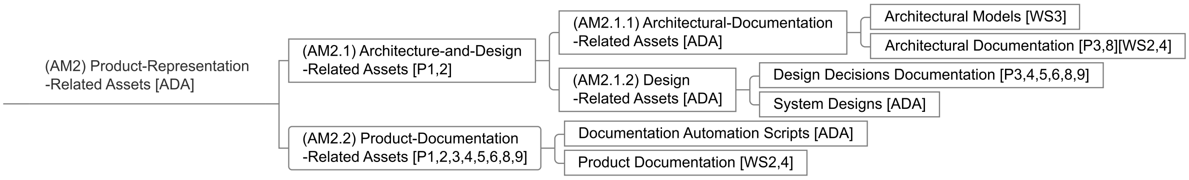

Fig. 6. Product-Representation-Related Assets Sub-tree.

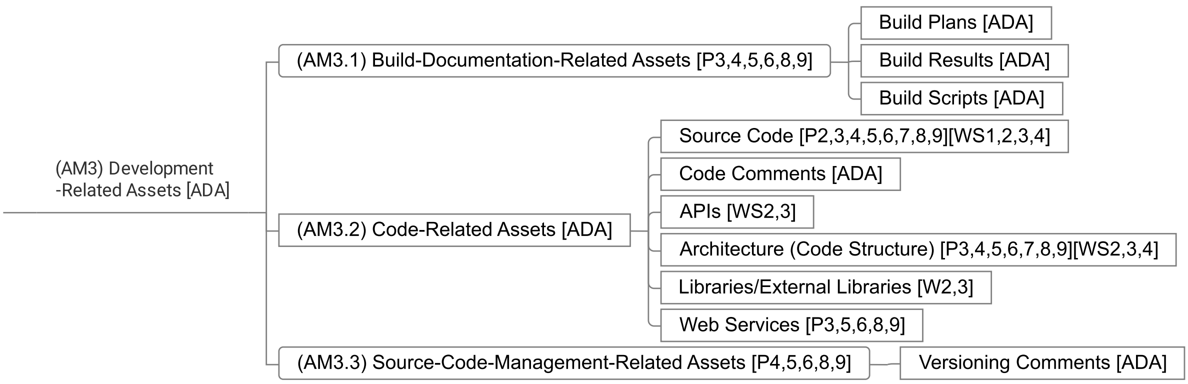

Fig. 7. Development-Related Assets Sub-tree.

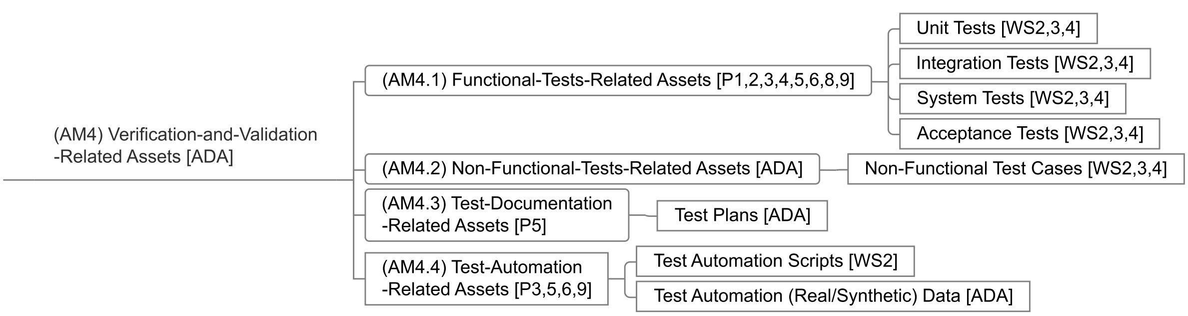

Fig. 8. Verification-and-Validation-Related Assets Sub-tree.

4.3.3. Development-Related Assets (AM3)

Development-Related Assets include the following three types of assets (see Fig. 7):

- *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 identified 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 identified 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 (Bass et al., 2003)—, 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 identified 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 Fig. 8):

- *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 identified the 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 identified *Non-Functional Test Cases* as an asset which belongs to this type.
- *Test-Documentation-Related Assets (AM4.3)* refer to the assets related to documenting the testing process. *Test Plans* is an asset we identified 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 identified 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 the use of the product or service (see Fig. 9). The

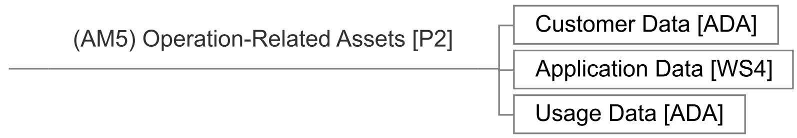

Fig. 9. Operation-Related Assets Sub-tree.

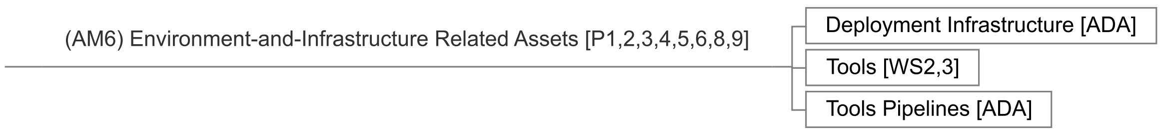

Fig. 10. Environment-and-Infrastructure-Related Assets Sub-tree.

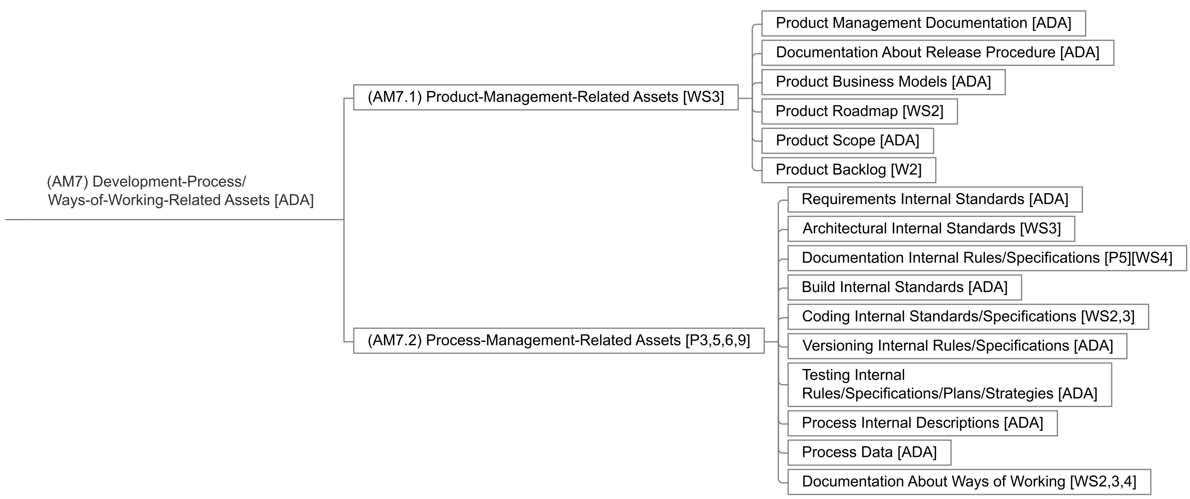

Fig. 11. Development-Process/Ways-of-Working-Related Assets Sub-tree.

operations-related assets include *Customer Data*, *Application Data*, and *Usage Data*.

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 Fig. 10). The environment-and-infrastructure-related assets include *Deployment Infrastructure*, *Tools*, and *Tools Pipelines*.

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 Fig. 11):

- *Product-Management-Related Assets (AM7.1)* refer to the assets related to the management of the product or service. These assets come from different stages, such as business justification, planning, development, verification, pricing, and product launching. We have identified 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 identified the following assets belonging to this type: *Requirements Internal Standards*, *Architectural 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. The identified organisation-related assets include Organisational Structure, Organisational Strategy, and Business Models (see Fig. 12):

4.4. Summary of validation workshops

In this section, we present the summary of the taxonomy validation workshop carried out with industrial practitioners, as described in Section 3.4.

The industrial participants were asked to select the assets from the taxonomy that they were not aware of or assets that were new to them. The participants collectively selected 33 assets. These assets and the number of times they were chosen are presented in Table 5. The participants were asked to select the assets that, based on their experience, should not be in the taxonomy.

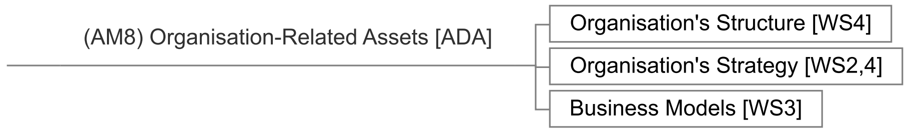

Fig. 12. Organisation-Related Assets Sub-tree.

Table 5
This table summarises the assets that the validation workshop participants were not aware of. The number represented the number of times the asset was selected by the participants.

	Asset	Selected by
1	**AM2.2** - Documentation Automation Scripts	4
2	**AM7.1** - Product Management Documentation	3
3	**AM1.2** - Quality Scenarios	2
4	**AM3.1** - Build Plans	2
5	**AM7.1** - Product Business Models	2
6	**AM7.2** - Requirements Internal Standards	2
7	**AM7.2** - Architectural Internal Standards	2
8	**AM7.2** - Documentation Internal Rules/Specifications	2
9	**AM7.2** - Build Internal Standards	2
10	**AM7.2** - Versioning Internal Rules/Specifications	2
11	**AM7.2** - Testing Internal Rules/Specifications/Plans/ Strategies	2
12	**AM7.2** - Process Internal Descriptions	2
13	**AM7.2** - Process Data	2
14	**AM1.**1 Feature-Related Backlog Items	1
15	**AM1.2** - User Experience Requirements	1
16	**AM2.1.1** - Architectural Models	1
17	**AM3.1** - Build Results	1
18	**AM3.2** - Architecture (Code Structure)	1
19	**AM3.2** - Libraries/External Libraries	1
20	**AM3.2** - Web Services	1
21	**AM3.3** - Versioning Comments	1
22	**AM4.1** - Acceptance Tests	1
23	**AM4.4** - Test Automation (Real/Synthetic) Data	1
24	**AM5** - Usage Data	1
25	**AM6** - Deployment Infrastructure	1
26	**AM6** - Tools	1
27	**AM7.1** - Documentation About Release Procedure	1
28	**AM7.1** - Product Scope	1
29	**AM7.1** - Product Backlog	1
30	**AM7.2** - Coding Internal Standards/Specifications	1
31	**AM7.2** - Documentation About Ways of Working	1
32	**AM8** - Organisation's Structure	1
33	**AM8** - Organisation's Strategy	1

Overall, the participants selected 13 assets. These assets and the number of times there were chosen are presented in Table 6.

In an open question, the participants were asked to write down the assets that were missing from the taxonomy. *Git Comments*, *Code Review Data*, *Contract Documentation*, and *Communication Channels* were mentioned in the questionnaire. Finally, we asked the participants to prioritise the top five assets based on their experience. The answers to the last question are summarised in Table 7.

The utility of the taxonomy is demonstrated through the use of taxonomy by practitioners and classification of existing knowledge (Britto et al., 2016). The participants in the validation workshops were able to use the taxonomy to identify and classify assets.

There are no existing taxonomies to compare the classification schemes; therefore, the taxonomy benchmarking was done with experts' knowledge. The validation workshop participants did not make any suggestions regarding the structure of the taxonomy.

Table 6
This table summarises the assets that the validation workshop participants, based on their experience, deemed not needed to be included in the taxonomy. The number represented the number of times the asset was selected by the participants.

	Asset	Selected by
1	**AM1.2** - Quality Scenarios	1
2	**AM3.1** - Build Plans	1
3	**AM3.2** - Libraries/External Libraries	1
4	**AM4.1** - Unit Tests	1
5	**AM5** - Usage Data	1
6	**AM7.1** - Product Management Documentation	1
7	**AM7.2** - Requirements Internal Standards	1
8	**AM7.2** - Architectural Internal Standards	1
9	**AM7.2** - Documentation Internal Rules/Specifications	1
10	**AM7.2** - Build Internal Standards	1
11	**AM7.2** - Versioning Internal Rules/Specifications	1
12	**AM7.2** - Testing Internal Rules/Specifications/Plans/ Strategies	1
13	**AM8** - Business Models	1

Table 7
This table summarises the assets that the validation workshop participants, based on their experience, selected as the top five assets. The number represented the number of times the asset was selected by the participants.

Asset	Selected by
AM1 - Product Requirements Related Assets	4
AM1.1 - Functional Requirements Related Assets	1
AM1.3 - Product Modification Related Assets	1
Use Cases	1
System Requirements	1
AM2 - Product Representation Related Assets	1
AM2.1.2 - Design Related Assets	2
Product Documentation	1
AM3 - Development Related Assets	3
AM3.2 - Code Related Assets	1
Source Code	1
Architecture	1
AM4 - Verification and Validation Related Assets	2
AM4.1 - Functional Tests Related Assets	1
AM5 - Operation Related Assets	1
AM6 - Environment and Infrastructure Related Assets	3
AM7 - Development Process/Ways of Working Related Assets	3
AM7.1 - Product Management Related Assets	1
AM8 - Organisation Related Assets	1

All participants agreed that the schema and categories (i.e., types of assets) were adequate.

The participants in the validation workshops selected the important assets based on their experience. The six most selected assets are *[AM1] - Product Requirements Related Assets*, *[AM3] - Development Related Assets*, *[AM6] - Environment and Infrastructure Related Assets*, *[AM7] - Development Process/Ways of Working Related Assets*, *[AM2.1.2] - Design Related Assets*, and *[AM4] - Verification and Validation Related Assets* (see Table 7). The selected important assets come from different asset types illustrating the importance of all assets. This emphasises the need to identify and maintain all assets in the organisation.

5. Discussion

This section discusses our findings in light of the research question, followed by the general lessons learned and implications.

5.1. Principal findings

RQ : *What assets are managed by organisations during the inception, planning, development, evolution, and maintenance of software-intensive products or services?*

In Section 4.3 we have presented a taxonomy of assets, which includes eight major types of assets *AM1* to *AM8*. Although the taxonomy is orthogonal by design (i.e., an asset or a type of asset can only be classified as a member of one type of assets), assets and types of assets are not isolated, i.e., some assets and types of assets are interrelated. For example, *architectural documentation* is directly related to *architecture* since architectural documentation is a representation of the architecture of the system.

During internal workshops and the taxonomy creation procedure, presented in Section 3.3, we identified several meta-characteristics of assets. These meta-characteristics 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**: Some assets are more prominent in industry and have been studied and discussed more deeply before, and therefore the asset is not alien anymore. For example, every software company, one way or another, has *Code*, in one form or another, or a *Product Backlog* with specific characteristics which is familiar to all people involved in the development of the software-intensive products or services.
- **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 or service.

Out of the eight major types of assets, two types have been studied more extensively, namely *Development-Related* Assets and *Product-Representation-Related* Assets. These results are not new and have been highlighted in previous studies such as Avgeriou et al. (2016), i.e., prior studies on TD 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 research articles. The reason behind this might be that:

- The TD metaphor was initially introduced in the context of prevalent assets (Avgeriou et al., 2016). Therefore the researchers have spent 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 does not align with certain "gold standards"; 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 (Ernst, 2012; Lenarduzzi and Fucci, 2019) and Process Debt (Martini et al., 2019). The TD metaphor was not initially used to deal with these types of assets (Avgeriou et al., 2016). 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 one can incur debt in software products. For example, incurring Documentation Debt might differ in different companies and development teams.

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.
- Some types of assets that are relevant to the industry have been understudied or not even studied at all. There is room for extending the research in such areas (Rios et al., 2018), e.g., *Operations-Related Assets (AM5)* and *Environment-and-Infrastructure-Related Assets (AM6)* (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.

Finally, our taxonomy of assets has an innate relationship with the TD research and the taxonomies, ontologies, secondary, and tertiary studies in the TD topic since as mentioned in Section 3.1, TD is one form of asset quality degradation. The taxonomy of assets is created using empirical evidence from peer-reviewed articles and co-production with industrial practitioners. The taxonomy of assets provides actual tangible assets, whereas other studies mainly present areas where assets belong to. Asset degradation goes beyond TD and considers different types of degradation, including the ripple effect and chain reactions of degradation that stems from the relations between assets. TD is a part of a bigger problem, i.e., managing asset degradation when companies deal with software-intensive products or services (Zabardast et al., 2022). And there is a need to consider a more holistic view to be able to address the asset degradation problem.

The synthesis of the results of taxonomy validation shows that the variety of processes followed by the participants' organisations is what makes some asset types well-known for some participants while seeming alien for others. The taxonomy can help widen practitioners' understanding regarding assets, by making some practitioners aware of assets that they are not used to.

Moreover, we observe that the assets in AM7 were selected more often than the other asset types (see Table 5). The results of the validation step solidify the need for the creation of the taxonomy.

5.2. Lessons learned

This section presents the lessons learned from running the industrial focus groups, 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 the industry, through these aspects have not received as much attention in the TD area (Avgeriou et al., 2016; Rios et al., 2018), although the TD community has already identified them as a research area that deserves more attention (Martini et al., 2019). Taking a look at some statements from participants in the industrial focus groups 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 environment. In the following, we will discuss the assets with similar characteristics and properties and assets that have implicit relations with 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 asset (e.g., Source Code) might impact or even imply the degradation of the other asset (e.g., Unit Tests) (Alégroth and Gonzalez-Huerta, 2017). 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 related assets. 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 defined and studied and those that lack a shared understanding or are underexplored. Therefore, the taxonomy provides a summary of the body of knowledge by linking empirical studies with industrial insights gathered through the industrial focus groups. Providing a common taxonomy and vocabulary:

- Makes it easier for the different 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 potential 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 to identify assets and the degradation that can impact other assets (ripple and chain effects that can spread the degradation to other assets like degradation of code causing degradation on test-code or vice-versa Alégroth and Gonzalez-Huerta, 2017), where such implicit degradation are not immediately detectable.

Finally, it might help large organisations to deal with developing software-intensive products or services that rely on external resources to help them achieve the business goals of their products. A major external contributor to new knowledge that can help 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 or services 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 findings to collect and organise the different assets and terminologies used to describe the assets will help practitioners be more aware of each type of asset and how they are managed in the context.

5.4. Limitations and threats to validity

In this section, we cover the limitations of our work and how they might affect the results. The taxonomy is created based on the data extracted from the literature review, the field study, and academic expert knowledge. We combined the inputs from the mentioned sources 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

(See Section 3.3). We encourage researchers and practitioners to consider the taxonomy within their organisation and identify the potential new types of assets 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 (2009) and Runeson et al. (2012).

Construct validity reflects the operational measurements and how the study represents what is being investigated. We are aware that the literature review is conducted in a limited area, i.e., TD. We chose the TD field as representative form of quality degradation since we believe TD is one form of asset quality degradation, as mentioned in Section 3.1. We acknowledge that the performed snowballing and limiting the literature review to a specific topic might affect the construct validity of this work since we could not cover all the topics and the articles related to those topics. However this threat is mitigated with the additional data collection from the focus groups.

We acknowledge that the participants of the focus groups do not include all the relevant stakeholders in organisations. We tried mitigating this threat by involving participants with different roles and varying expertise from the companies. We are also aware that the participants' statements in the focus groups can be interpreted differently by the researcher and the participants. We mitigate this threat in three ways. First, by sending the summary, with the transcription of their statements and our own notes, back to the participants, asking for their feedback; second, by having two researchers code the raw data independently; and third, 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. We have provided a systematic way of extending the taxonomy, i.e., the meta-model. 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 the involvement of all the roles in these organisations.

Reliability refers to the extent that the data and analysis are dependent on the researchers. When conducting qualitative studies, the goal is to provide results that are consistent with the collected data (Merriam and Tisdell, 2015). We have tried to mitigate this threat, i.e., consistency of the results. Firstly, by rigorously documenting and following the procedures of the focus groups to collect consistent data (Yin, 2009). And secondly, by relying on consistency during the analysis, i.e., blind labelling of the data by multiple researchers and peer reviewing the labels.

6. Conclusions and future work

This paper presents a taxonomy for classifying assets with inherent value for an organisation subject to degradation. These assets are used during the development of software-intensive products or services. The creation of the taxonomy of assets attempts to provide an overarching perspective on various assets for researchers and practitioners. The taxonomy allows us to characterise and organise the body of knowledge by providing a common vocabulary of and for assets.

Eight major types of assets 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 spread the body of knowledge.

The taxonomy helps draw out the assets with similar characteristics and implicit relations among them. Most of such similarities of characteristics, properties, and relations are not immediately visible when considering the assets from only one perspective. Taking a more abstract and high-level look at the assets involved in the development of software-intensive products or services can help facilitate the management activities and the overall development process.

The dimensions provided by our taxonomy are not exhaustive, nor are the assets we identified. Therefore, we intend to conduct further investigation to complement the taxonomy by incorporating the new knowledge. Furthermore, we would like to study the relationship between assets, how the degradation of an asset can have a ripple effect and chain reactions, causing the degradation of other assets, and how the degradation impacts the development process. Lastly, we intend to investigate the individual properties of assets to identify the metrics used for measuring assets, their value, and their degradation (or lack thereof).

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 its different types.

CRediT authorship contribution statement

Ehsan Zabardast: Conceptualization, Methodology, Formal analysis, Investigation, Data curation, Writing – original draft. **Javier Gonzalez-Huerta:** Conceptualization, Methodology, Investigation, Data curation, Writing – review & editing, Project administration. **Tony Gorschek:** Conceptualization, Writing – review & editing, Supervision. **Darja Šmite:** Formal analysis, Data curation, Writing – review & editing. **Emil Alégroth:** Investigation, Writing – review & editing. **Fabian Fagerholm:** Investigation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Appendix. The asset management taxonomy

Assets' definitions are presented in this section in Tables 8 to 15. The full tree of the asset management taxonomy is presented in Fig. 13.

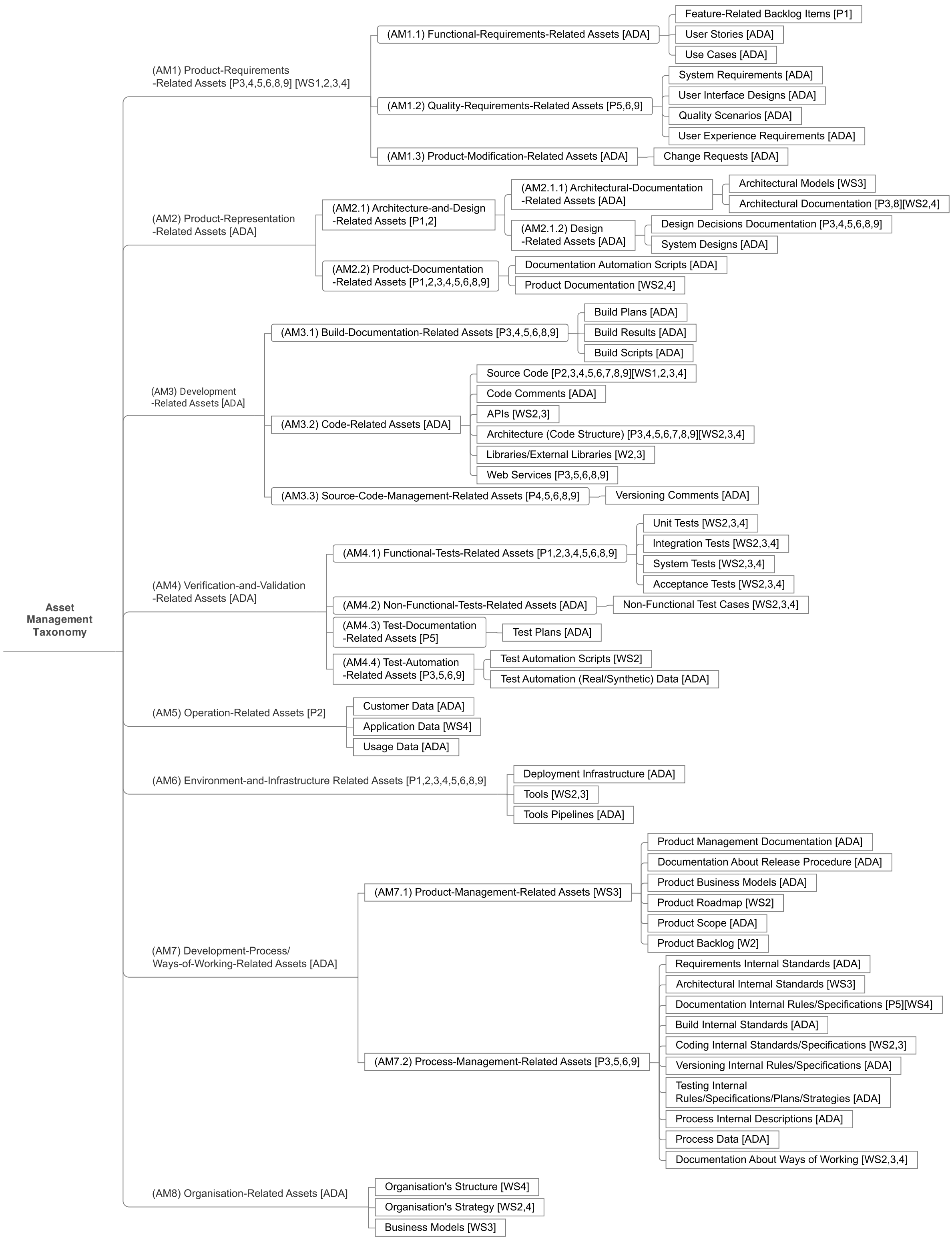

Fig. 13. The asset management taxonomy.

Table 8
Product-Requirements-Related Assets are listed in this table. The table contains the definitions of the assets and their type.

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 (Müter et al., 2019).
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 (Müter et al., 2019).
Use Cases	AM1.1	Use Cases are lists of actions or events that describe how a user will achieve a goal in a system (Kruchten, 2004).
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 (Bourque et al., 2014)".
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 (Bourque et al., 2014)".
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 (Bass et al., 2003)". A quality scenario is a way of stating a requirement in an unambiguous and testable manner (Bass et al., 2003).
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 (Law and Van Schaik, 2010; Kujala and Miron-Shatz, 2013)".
Change Requests	AM1.3	Change Requests are the modifications to the software product that are not coming from the requirements analysis of the product.

Table 9
Product-Representation-Related Assets are listed in this table. The table contains the definitions of the assets and their type.

Asset	AM type	Definition
Architectural Models	AM2.1.1	Architecture Models are partial abstractions of systems, they capture different properties of the system (Kruchten, 1995). "Architecture modelling 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 (Kumar et al., 2014)".
Architectural Documentation	AM2.1.1	Architectural Documentation are the representations of the decisions made to construct the architecture of the software (Kruchten, 1995).
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 (Bourque et al., 2014).
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.

Table 10
Development-Related Assets are listed in this table. The table contains the definitions of the assets and their type.

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 (Kernighan, 1974).
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 (Grubb and Takang, 2003).
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 (Bass et al., 2003).

(*continued on next page*)

Table 10 (*continued*).

Asset	AM type	Definition
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.

Table 11
Verification-and-Validation-Related Assets are listed in this table. The table contains the definitions of the assets and their type.

Asset	AM type	Definition
Unit Tests	AM4.1	Unit Tests are the tests written to examine the individual units of the code (Hamill, 2004). "Unit tests generally focus on the program logic within a software component and on correct implementation of the component interface (Berner et al., 2005)".
Integration Tests	AM4.1	Integration Tests are the tests written to examine the combined set of modules as a group (Berner et al., 2005; ISO/IEC/IEEE, 2010).
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.

Table 12
Operations-Related Assets are listed in this table. The table contains the definitions of the assets and their type.

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.

Table 13
Environment-and-Infrastructure-Related Assets are listed in this table. The table contains the definitions of the assets and their type.

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.

Table 14
Development-Process/Ways-of-Working-Related Assets are listed in this table. The table contains the definitions of the assets and their type.

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.

(*continued on next page*)

Table 14 (*continued*).

Asset	AM type	Definition
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.

Table 15
Organisation-Related Assets are listed in this table. The table contains the definitions of the assets and their type.

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.

References

Alégroth, E., Gonzalez-Huerta, J., 2017. Towards a Mapping of Software Technical Debt onto Testware. In: 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications. Vienna, Austria, pp. 404–411, URL: http://dx.doi.org/10.1109/SEAA.2017.65.

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. Inf. Softw. Technol. 70, 100–121.

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.

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.

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.

Badampudi, D., Wohlin, C., Petersen, K., 2015. Experiences from using snowballing and database searches in systematic literature studies. In: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. pp. 1–10.

Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice. Addison-Wesley Professional.

BenIdris, M., 2020. Investigate, identify and estimate the technical debt: a systematic mapping study. Int. J. Softw. Eng. Appl. (IJSEA) 9 (5).

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. http://dx.doi.org/10.1109/ICSE.2005.1553603.

Besker, T., Martini, A., Bosch, J., 2017. Time to pay up: Technical debt from a software quality perspective. In: CIbSE. pp. 235–248.

Besker, T., Martini, A., Bosch, J., 2018. Managing architectural technical debt: A unified model and systematic literature review. J. Syst. Softw. 135, 1–16.

Blum, B.I., 1994. A taxonomy of software development methods. Commun. ACM 37 (11), 82–94.

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.

Britto, R., Wohlin, C., Mendes, E., 2016. An extended global software engineering taxonomy. J. Softw. Eng. Res. Dev. 4 (1), 1–24.

Broughton, V., 2015. Essential Classification. Facet Publishing.

Broy, M., 2018. A logical approach to systems engineering artifacts: semantic relationships and dependencies beyond traceability—from requirements to functional and architectural views. Softw. Syst. Model. 17 (2), 365–393.

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.

Coghlan, D., Brannick, T., 2014. Doing Action Research in Your Own Organization, fourth ed. Sage, London.

Constantopoulos, P., Doerr, M., 1995. Component classification in the software information base. Object-Oriented Softw. Compos. 177.

Cunningham, W., 1992. The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger 4 (2), 29–30.

Ernst, N.A., 2012. On the role of requirements in understanding and managing technical debt. In: 2012 Third International Workshop on Managing Technical Debt. MTD, pp. 61–64. http://dx.doi.org/10.1109/MTD.2012.6226002.

Fox, M., Green, G., Martin, P., 2007. Doing Practitioner Research. Sage.

Garriga, M., 2017. Towards a taxonomy of microservices architectures. In: International Conference on Software Engineering and Formal Methods. Springer, pp. 203–218.

Glass, R.L., Vessey, I., 1995. Contemporary application-domain taxonomies. IEEE Softw. 12 (4), 63–76.

Glass, R.L., Vessey, I., Ramesh, V., 2002. Research in software engineering: an analysis of the literature. Inf. Softw. Technol. 44 (8), 491–506.
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.
Grubb, P., Takang, A.A., 2003. Software Maintenance: Concepts and Practice. World Scientific.
Hamill, P., 2004. Unit Test Frameworks: Tools for High-Quality Software Development. O'Reilly Media, Inc..
Idowu, S., Strüber, D., Berger, T., 2022. Asset management in machine learning: State-of-research and state-of-practice. ACM Comput. Surv. http://dx.doi.org/10.1145/3543847.
ISO/IEC/IEEE, 2010. Systems and Software Engineering ISO/IEC/IEEE 24765:2010. Technical Report, ISO/IEC/IEEE.
Kernighan, B.W., 1974. Programming in C- A Tutorial. Bell Laboratories, Unpublished Internal Memorandum.
Klotins, E., Unterkalmsteiner, M., Gorschek, T., 2018. Software-intensive product engineering in start-ups: a taxonomy. IEEE Softw. 35 (4), 44–52.
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.
Kruchten, P.B., 1995. The 4+ 1 view model of architecture. IEEE Softw. 12 (6), 42–50.
Kruchten, P., 2000. The rational unified process 2nd edition: an introduction.
Kruchten, P., 2004. The Rational Unified Process: An Introduction. Addison-Wesley Professional.
Kruchten, P., Nord, R.L., Ozkaya, I., 2012. Technical debt: From metaphor to theory and practice. IEEE Softw. 29 (6), 18–21.
Kruchten, P., Nord, R., Ozkaya, I., 2019. Managing Technical Debt: Reducing Friction in Software Development. Addison-Wesley Professional.
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.
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.
Kwasnik, B.H., 1992. The role of classification structures in reflecting and building theory. Adv. Classif. Res. Online 3 (1), 63–82.
Law, E.L.-C., Van Schaik, P., 2010. Modelling user experience–An agenda for research and practice. Interact. Comput. 22 (5), 313–322.
Lehman, M.M., 1979. On understanding laws, evolution, and conservation in the large-program life cycle. J. Syst. Softw. 1, 213–221.
Lehman, M., 1996. Laws of software evolution revisited. In: European Workshop on Software Process Technology. Springer, Berlin, Heidelberg, pp. 108–124.
Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., Arcelli Fontana, F., 2021. A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. J. Syst. Softw. 171, 110827. http://dx.doi.org/10.1016/j.jss.2020.110827, URL: https://www.sciencedirect.com/science/article/pii/S016412122030220X.
Lenarduzzi, V., Fucci, D., 2019. Towards a holistic definition of requirements debt. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–5. http://dx.doi.org/10.1109/ESEM.2019.8870159.
Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw. 101, 193–220.
von Linné, C., 1735. Systema Naturae; Sive, Regna Tria Naturae: Systematice Proposita Per Classes, Ordines, Genera & Species. Haak.
Martini, A., Stray, V., Moe, N.B., 2019. Technical-, social- and process debt in large-scale agile: An exploratory case-study. In: Hoda, R. (Ed.), Agile Processes in Software Engineering and Extreme Programming – Workshops. Springer International Publishing, Cham, pp. 112–119.
Méndez, D., Böhm, W., Vogelsang, A., Mund, J., Broy, M., Kuhrmann, M., Weyer, T., 2019. Artefacts in software engineering: a fundamental positioning. Softw. Syst. Model. 18 (5), 2777–2786.
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.
Merriam, S.B., Tisdell, E.J., 2015. Qualitative Research: A Guide to Design and Implementation. John Wiley & Sons.
Miles, M.B., Huberman, A.M., Saldaña, J., 2014. Qualitative data analysis: A methods sourcebook. 3rd.
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.
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.
Reussner, R., Goedicke, M., Hasselbring, W., Vogel-Heuser, B., Keim, J., Märtin, L., 2019. Managed Software Evolution. Springer Nature.
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. Inf. Softw. Technol. 102, 117–145.
Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131–164.
Runeson, P., Host, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons.
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.
Saldaña, J., 2015. The Coding Manual for Qualitative Researchers. Sage.
Silva, M., Oliveira, T., Bastos, R., 2009. Software artifact metamodel. In: 2009 XXIII Brazilian Symposium on Software Engineering. pp. 176–186. http://dx.doi.org/10.1109/SBES.2009.28.
Šmite, D., Wohlin, C., Galviņa, Z., Prikladnicki, R., 2014. An empirically based terminology and taxonomy for global software engineering. Empir. Softw. Eng. 19 (1), 105–153.
Sommerville, I., 2015. Software engineering. 10th. In: Book Software Engineering. 10th, Series Software Engineering. Addison-Wesley.
Stol, K.-J., Fitzgerald, B., 2018. The ABC of software engineering research. ACM Trans. Softw. Eng. Methodol. 27, 1–51. http://dx.doi.org/10.1145/3241743.
Stringer, E.T., 2014. Action Research, fourth ed. Sage, Thousand Oaks, CA.
Svahnberg, M., Van Gurp, J., Bosch, J., 2005. A taxonomy of variability realization techniques. Softw. - Pract. Exp. 35 (8), 705–754.
Taivalsaari, A., Mikkonen, T., 2018. A taxonomy of IoT client architectures. IEEE Softw. 35 (3), 83–88.
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.
Tom, E., Aurum, A., Vidgen, R., 2012. A consolidated understanding of technical debt. In: ECIS 2012 Proceedings.
Tom, E., Aurum, A., Vidgen, R., 2013. An exploration of technical debt. J. Syst. Softw. 86 (6), 1498–1516.
Unterkalmsteiner, M., Feldt, R., Gorschek, T., 2014. A taxonomy for requirements engineering and software test alignment. ACM Trans. Softw. Eng. Methodol. (TOSEM) 23 (2), 1–38.
Usman, M., Britto, R., Börstler, J., Mendes, E., 2017. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. Inf. Softw. Technol. 85, 43–59.
Wohlin, C., 2014. 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.
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.
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.
Yin, R.K., 2009. Robert k. Yin. Case Study Res.: Design and Methods 4.
Zabardast, E., Frattini, J., Gonzalez-Huerta, J., Mendez, D., Gorschek, T., Wnuk, K., 2022. Assets in software engineering: What are they after all? J. Syst. Softw. 193, 111485. http://dx.doi.org/10.1016/j.jss.2022.111485, URL: https://www.sciencedirect.com/science/article/pii/S0164121222001662.
Zhao, Y., Dong, J., Peng, T., 2009. Ontology classification for semantic-web-based software engineering. IEEE Trans. Serv. Comput. 2 (4), 303–317.

Ehsan Zabardast is a Ph.D. candidate of Software Engineering in Software Engineering Research Lab Sweden at Blekinge Institute of Technology. He holds a master's degree in informatics and data science. His main research involves software assets, asset management and degradation, technical debt, and software architecture. His current work includes studying how assets degrade, considering other aspects of software development. A major part of his research involves studying technical debt and growing out of the metaphor.

Javier Gonzalez-Huerta is an associate professor in the Software Engineering Department at BTH. He received his Ph.D. in Computer Science from the Universitat Politècnica de Valencia (UPV) in 2014, after working in the industry for about 15 years. Javier's research focuses on the Asset Management and Technical Debt areas, and he has been doing applied research together with the SERT industrial partners for more than five years.

Prof. Dr. Tony Gorschek is a Professor of Software Engineering at Blekinge Institute of Technology — where he works as a research leader and scientist in close collaboration with industrial partners. Currently he is leading the S.E.R.T.

research profile, the PLEng industrial research school, the Ericsson SPACE lab for software engineering, and several other industrially applied research projects. Dr. Gorschek has over fifteen years industrial experience as a CTO, senior executive consultant and engineer.

Darja Smite is a Professor of Software Engineering at Blekinge Institute of Technology and a part-time research scientist at SINTEF ICT. She has dedicatedly focused on understanding the impact of globalisation and offshoring in software companies. Smite has conducted research with and for a number of international companies such as Ericsson, ABB, Emerson Process Management, Boss Media, CALVI and DXC, and has insights from cooperating with offshore vendors in India, China, Poland, Latvia, Ukraine and Russia.

Dr. Emil Alégroth is a researcher at Blekinge Institute of Technology. His main research area is software verification and validation with specialist interest in graphical user interface (GUI) testing. This research has been conducted mostly in co-production with software industry, including companies like Spotify, Saab, Jeppesen, Ericsson, AddQ, Handelsbanken, Swedbank, and more. Whilst much of this research has been focused on Software Engineering, Emil has also conducted cross-disciplinary research within Psychology (How human-factors such as team-cohesion and stress influence testing), Didactics (How student knowledge/experience of online platforms influence their learning) and Economy (How to model the cost and alternative costs of creating/maintaining/removing tests). In addition to his academic background, Emil is also operational in a couple of start-up companies and the non-profit organisation SAST (Swedish Association for Software Testing) that hosts free conferences on testing to Swedish test professionals.

Fabian Fagerholm is associate senior lecturer at Blekinge Institute of Technology. His research interests include developer experience, human and behavioural aspects of software engineering, continuous experimentation and evidence-driven software product development, open source software development, and experiential and project-based software engineering education. At Blekinge Institute of Technology, his teaching is related to behavioural and human aspects of software engineering, and Lean and Agile software processes. He received his Ph.D. in computer science from the University of Helsinki.