The Journal of Systems & Software 190 (2022) 111336

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

New Trends and Ideas

Infinite technical debt☆

Melina Vidoni [a,*], Zadia Codabux [b], Fatemeh H. Fard [c]

[a] Australian National University, Australia
[b] University of Saskatchewan, Canada
[c] University of British Columbia, Canada

ARTICLE INFO

Article history:
Received 3 February 2022
Received in revised form 8 April 2022
Accepted 13 April 2022
Available online 22 April 2022

Keywords:
Technical debt
Game theory
Software lifecycles
Software maintenance and evolution

ABSTRACT

Code ridden with Technical Debt (TD) has motivated software engineers to keep the quality of systems under control to ease future maintenance tasks. In the last decade, there have been significant advances regarding TD management (TDM). However, research about incorporating TDM into the software development lifecycle remains scarce, and existing approaches aim to control TD through different processes. This proposal leverages the concept of infinite games from game theory to posit a different perspective. We argue that TD cannot be entirely removed and that its effects or consequences cannot be considered "managed" even when an occurrence (i.e., a smell) is repaid. Rather than using a mathematical approach, we present TDM in terms of the four components of infinite games (players, rules, goals, and time), its tradeoffs and relationships, to discuss its potential impact on TDM activities. As this is an incipient area, our goal is to motivate a change of mindset regarding TDM, stimulating reflective thinking and thus, posing a new line of research. We conclude with a series of potential research questions organised into three key areas.

Contents

☆ Editor: Neil Ernst.

* Corresponding author.
E-mail addresses: melina.vidoni@anu.edu.au (M. Vidoni), zadiacodabux@ieee.org (Z. Codabux), fatemeh.fard@ubc.ca (F.H. Fard).

https://doi.org/10.1016/j.jss.2022.111336
0164-1212/

1. Introduction

Technical Debt (TD) refers to the consequences of taking shortcuts and making poor design choices during the development or maintenance of a software system, negatively impacting its future evolution (Nielsen et al., 2020). Prior work has uncovered multiple types of TD which can be identified using multiple indicators, but sharing mostly similar management strategies (Alves et al., 2016). TD Management (TDM) is described as "the actions of identification, assessment, and remediation of technical debt in a software system" (Griffith et al., 2014). Studies consolidating the findings have created unified activities that abstract the specificities of each TD type, to consider TDM as an overarching framework (Alfayez et al., 2020; Lenarduzzi et al., 2021).

Therefore, TDM is crucial because it enables an organisation to have a more optimal use of its resources; when the right TDM techniques are applied, the development team can identify technical debt and repay it in a timely fashion (Alves et al., 2016). Moreover, the importance of managing TD embedded in software products has been highlighted by recent studies (Li et al., 2015; Ramasubbu and Kemerer, 2019). TDM facilitates decision-making concerning the need to eliminate a TD item and provides guidance on the most appropriate time to do so (Seaman and Guo, 2011), allowing a reduction of TD negative impact, and is "a decisive factor for the success of software projects" (Freire et al., 2020). If not controlled, TD can cause financial and technical problems, such as "increasing software maintenance and evolution costs, leading to a crisis point where the future of the software is jeopardised" (Rios et al., 2018).

In the past decade, most TD studies focused on identification, but the broader management (namely, TDM and its related sub-areas) remained understudied (Nielsen et al., 2020; Malakuti and Ostroumov, 2020; Schütz and Gómez, 2020; Rios et al., 2021). In particular, "the lack of comprehensive solutions that consider a management process for TD as a whole can make debt management difficult to perform" (Rios et al., 2018). This has led to several repercussions in the industry, including startups facing obstacles to implement TDM (Besker et al., 2018).

To the best of our knowledge, most studies have approached TDM with a similar mindset: that it is possible to remove TD entirely from the development lifecycle (Schütz and Gómez, 2020; Pujols et al., 2020; Rios et al., 2021; Malakuti and Ostroumov, 2020; Nielsen et al., 2020; Buchmann and Haki, 2021; Martini et al., 2018). However, TD cannot be entirely eliminated from a project except in special circumstances (e.g., the project retires) (Krishna and Basu, 2012; Kruchten et al., 2013). As a result, this position paper proposes to change the perspective of TDM from expecting to altogether remove and control TD to a process that never ends. This idea is based on the concept of *infinite games* discussed in game theory and often adopted (albeit loosely) for managerial purposes. In particular, an *infinite game* is a process with a large number of choices, where the goal is to continue playing for an indefinitely long period (D'Alotto, 2020; Le Roux and Pauly, 2020), that was originally proposed to "reflect on the nature of play itself" without the use of advanced mathematics[1]

The rationale for selecting *infinite games* lies in the changeable nature of software and its ability to endure and evolve, known as *software sustainability*, and the fact that software is multi-version, multi-people, and multi-uses (Venters et al., 2018). Prior works demonstrated that approximately 50%–70% of a system's total lifecycle cost is spent on its evolution and maintenance (Garcia et al., 2013; Taivalsaari and Mikkonen, 2017), given that a software's sustainability is concerned with its availability, extensibility (to grow over time), and maintainability (to be repaired and enhanced over time) (Venters et al., 2014). Therefore, "software maintenance" is an evolving infinite game, "played" to make a piece of software "survive" the longest. Developers add their revisions, modify code, and release new versions. However, TD cannot be resolved as either (a) the actions themselves may bring (willingly or unwillingly) more TD, (b) the underlying software is constantly changing, or (c) the environmental changes cause TD to incur in the software. Hence, TD (regardless of its type) cannot be removed, being as infinite and unending as the software lifecycle. The only option to finally "end" TD is when the software retires (i.e., when all players stop playing).

Although similar approaches have been assessed from a purely mathematical perspective to solve and optimise algorithms (Le Roux and Pauly, 2020; Thomas, 2002), this is not the goal of this position paper. The latter does not contribute to the advanced mathematics field, nor produces mathematical models to represent TDM as a game-theoretic model but approaches the concept of *infinite games* for *infinite technical debt* more broadly, to stimulate reflective thinking and promote a novel and different line of action and research among practitioners and academics, both present and future.

2. Related works

Many studies have approached a number of software development areas from a game-theoretic perspective.

Skourletopoulos et al. (2017b) developed a game-theoretical approach for TD analysis in terms of quality of service and experience for cloud-based systems. They developed a purely mathematical theory based on the Nash equilibrium and proved it as a theorem through mathematical formulations but without any application in practice. Overall, their proposal provided a theoretical organisation to achieve resource leasing and change optimisation on cloud service level using TD analysis in terms of quality of service and quality of experience. Other works on cloud-based systems worked alongside this approach, extending its reach (Kokol et al., 2021; Skourletopoulos et al., 2017a).

Bavota et al. (2010) used game theory techniques to automatically recommend A specific type of refactoring (namely, extract-class). The authors provided a thorough mathematical formulation and performed a preliminary evaluation with data from open-source systems. Though results were positive, the proposal was limited to a single type of refactoring and tested in a controlled environment.

Regarding mathematical approaches, other authors conducted theoretical studies with empirical validations in many sub-areas. Examples are decision-making for self-adaptive software applied to the internet of things (Lee et al., 2019), requirements change management for green computing (Tong et al., 2017), automated verification of robustness in distributed systems (Fernando, 2018), and optimisation of software crowd-sourcing (Hu and Wu, 2014).

Gavidia-Calderon et al. (2020) performed a survey of game theory studies in different areas of software engineering and determined that most studies are "highly idealised models that are rarely based on process data." Moreover, they argue that the reason for this is that game theory analysis needs to cope with exponentially large trees (of options), which restrains the analysis. However, they investigated the all-encompassing area of software engineering in a generalised approach. They concluded that the possibilities of using game theory to improve software processes are boundless and should be leveraged and finished their work with a community call to do so. In terms of Systems-of-Systems (SoS), Axelsson (2019) conducted a literature review and concluded that game theory has often been used to model

[1] https://tinyurl.com/3mpksu7x. As a result, this paper also aims to reflect on the nature of TD itself, and subsequently, propose a different play (TDM).

SoS mechanisms. They affirmed that "most results in the field lack validation in practice", regardless of game theory being well suited to deal with problems related to acquisition, design, and operations.

Other disciplines have welcomed research regarding infinite planning horizons through game-theoretic approaches, as a new perspective in their field. One case is incorporating evolutionary game theory into multiple disciplines (Sandholm, 2020), such as economics and strategic management (Ozkan-Canbolat et al., 2016). Those frameworks have also been adjusted to other branches of computer science. For example, to manage information-sharing about cyberattacks to achieve stability (Tosh et al., 2015), discovering stable network routes for data dissemination (Khan et al., 2018), incentive mechanisms in peer-to-peer networks (Cui et al., 2015), and assess and determine materialised views in data warehouses design (Sohrabi and Azgomi, 2019).

Novelty. This position paper is putting forward a novel approach for the TD domain; namely, the use of game theory from a managerial perspective to propose a *change of mindset* regarding how to deal and approach TDM in the software development process. We do not advocate for specific mathematical optimisations and instead focus on a single process: the management of TD throughout a software's lifecycle to ensure software sustainability. As a result, the main limitation of this position is the ability to test and evaluate it. Though there had been some proposals providing tools and checkpoints to assess the lifecycle, the most honest and trustworthy evaluation only comes from repeated applications (Hesari et al., 2010). This paper is thus a call for the community to test and apply the proposed ideas in practice.

3. TDM: An infinite game

There are two types of games: finite and infinite (Carse, 2011).

Finite games have known players, fixed rules, and an established objective that, when achieved, allows players to win and lose; here, the players freely choose to play, and the rules are clear (Weber, 2020). These *finite games* can have *perfect information*, when all the moves and positions of players are known to the other players all the time, or imperfect games (e.g., those games with *imperfect information*) when this does not happen. Game theory states that finite games with a clear winning condition and perfect information are determined (e.g., their winning condition can be achieved in a finite number of steps) (Dziembowski et al., 1997). In terms of TDM, this could be considered as a baseline case in which whenever new TD is introduced, (a) all players are automatically and accurately informed, (b) they all agree to remove the TD occurrence (regardless of its type), (c) they do so immediately, and (d) therefore, there are no consequences caused by that TD instance. This is not a realistic approach, even when using automated detection/correction tools, given that it is known there are no infallible solutions in software engineering (Bessey et al., 2010), and a piece of software is part of a system that involves people which, regardless of the established processes, will subjectively appraise the situation (Calp and Akcayol, 2015).

Infinite games have rules that only exist to continue playing (Weber, 2020). *Infinite games* with perfect information still have achievable winning strategies (derived from a winning condition), a strict alternation of moves between players; perfect infinite games are often represented in graphs and played as automata, since they are still considered decidable (Holtmann et al., 2010). In this specific setting, the players must be informed of every move or event (Berwanger and Doyen, 2020). However, possessing such detailed information is often unrealistic in terms of TDM, given that even the most state-of-the-art tools for TD detection only are not fully accurate, meaning they cannot detect all cases of TD (Lenarduzzi et al., 2021); even then, the players may decide to maintain the TD (namely, not repay it) for a given reason (Ramasubbu and Kemerer, 2019; Malakuti and Ostroumov, 2020).

However, "infinite games where several players seek to coordinate under imperfect information are deemed to be undecidable, unless the information is hierarchically ordered among the players" (Berwanger and Mathew, 2017). Since each play of an infinite game can change the game's boundaries and extend it, it is not possible to know for how long that game has been played or when it will end (thus, related to the software sustainability Venters et al., 2018). Hence, players are "incentivised to adjust their actions, reconfigure boundaries, share power, and negotiate terms of engagement for the sole purpose of keeping the game alive" (Carse, 2011). This situation more accurately aligns with TDM given that there is no strict alternation of players (namely, they may even be playing simultaneously), and both the goals and the rules of the game may change (e.g., higher quality suddenly sacrificed to meet a deadline with all features; the budget is adjusted).

To study TDM as an infinite game, we define the management process in terms of four elements: time, rules, players, and goals. These four elements are highly interrelated, and although an abstract definition can be given here, they need to be further specified for each particular, individual software development process. The following subsections will discuss them; Note that these elements are intrinsically related, and presenting them in a linear manner is not ideal, but the best approach to convey our ideas for this paper.

3.1. Time

Any given finite game can be played repeatedly, but each occurrence is unique, and depends on the combination of the four elements; as a result, it cannot be fully replicated (Weber, 2020). Overall, infinite game has no predetermined beginning, middle or end, but may have "finite outcomes" acting as internal "milestones" or situations that must be solved in order for the play to continue (Holtmann et al., 2010). These often manifest as "time markers" within the game. "Time markers" are specific points in time when the players assess their performance, but are not opportunities to "end the game" (Carse, 2011). The lack of an opportunity to "end the game" is due to its condition of being "infinite" and thus "undecided" (Berwanger and Mathew, 2017). *Note*: it is possible for some or all Players to enforce the game to end, discussed after approaching their interaction with Goals and Rules.

By definition, the TD metaphor stresses the short-term gain given by a sub-optimal solution compared to a long-term one that is considered more appropriate and less damaging (Nielsen et al., 2020). However, the "game" or "play" (i.e., TDM) continues while the software is planned, developed, used, and maintained, throughout the entire software development lifecycle, from its inception to its "demise" (retirement).

The aforementioned "time markers" in TDM and software development can be sprints, cycles, and iterations (i.e., the next upcoming milestone) determined by the lifecycle process being followed. However, they can also be events (detected or imposed) that must be handled for the play to continue. In either case, these markers will have an effect on the Goals and Rules (either temporarily or not), as the Players attempt to handle the marker. This can become a "feedback loop" because to decide whether that "time marker" has been successfully completed, the Players will use other Rules to assess the outcomes—namely, measurements taken to evaluate the current state of the software, budget availability, time and effort availability, market analysis, among others (Becker et al., 2018).

For example, during a sprint review (i.e., a time marker), a manager (player) can bring a negative market analysis and a new budget limitation (rules) to the development team (also players). This will cause changes on the next iteration, as the team may decide to "change the goals" and postpone some features to introduce new changes that improve the user's perception (users are players). This will bring a new set of rules, that could be measuring user satisfaction, equivalent or complementary to validation testing.

3.2. Players

Several points can be discussed about Players.

Main Players. In TDM, the Players are all the groups of stakeholders' related to a software. Hence, the developers, testers, users, managers with any decision power over the software, and even other systems (Lenarduzzi et al., 2021; Alfayez et al., 2020).

Additionally, the "uncontrollable environment" (namely, the environment not controlled by the developers/stakeholders, other software such as operative systems) would be what game theory frameworks label as "nature" (Berwanger and Mathew, 2017)– this represents external elements that may affect the software. Related software (i.e., software related to the software in play) mutates, and the context changes as society evolves (Garcia et al., 2013; Venters et al., 2014). For example, operating systems, the laws of the country where the software is used and deployed (e.g., Europe's GDPR forced many systems to update (Laybats and Davies, 2018)), social considerations altering systems (e.g., GitHub's renaming of the "master" branch[2]).

Play History. The system itself is related to the players because it reflects the "play history" (namely, the sequence of actions and plays done by the players). Though there are means to trace the actions (e.g., version control systems, bug issues, planning tools) (Rubasinghe et al., 2018), including those specific to TDM (Alves et al., 2016; Zazworka et al., 2013), it remains unfeasible to communicate every detail as detection tools are continuously being improved, and particularly, more established for some TD types than others (Ramasubbu and Kemerer, 2018).

Note: Achieving a full traceable, complete, instantly updated play history, with *all* players being immediately and thoroughly informed of the change could move TDM from being an *infinite game with imperfect information* to one of *perfect information*, thus simplifying the stakes.

Trade-offs in Plays. A consideration in infinite games is that: (i) the players play against each other (Weber, 2020), and (ii) their joint actions determine the "play" (game) outcome, because (iii) there is no alternation of plays and all could be acting simultaneously (Berwanger and Mathew, 2017). In particular:

- Examples of (i): A user will not use the system as intended. Developers may self-sabotage themselves by incurring TD (i.e., this makes future work on the software harder than it should be) (Alves et al., 2016). Managers may affect the process by reducing the allocated budget. Nature (the environment) may add TD to a software (e.g., an operating system update, or a new law that requires updating the software).
- Example of (ii): if users suddenly favour one feature that was a secondary add-on, their value in the system will change and in order to continue playing, the other stakeholders must adjust. An example of this is how Slack (a communication platform) started as a gaming company[3]; this is often referred to as an "ideal" in terms of infinite games[1].
- Example of (iii): the users may be using version 1.2 of the system, while the developers work on a patch, and the government decides a change on privacy laws that will affect future versions.

Forcing an End. Albeit infinite games continue, players can decide to stop playing when they run out of resources (Carse, 2011). Therefore, players do not play for themselves, but to continue the game (Weber, 2020) (to profit from the software). *Note:* the word "profit" is used loosely, and its meaning will depend on the type of Players. For example, while a regular user may play a mobile game for fun, a developer implements it for an economic gain.

Therefore, "leaving players out" refers to situations where the cost of managing and removing TD increases to the point that maintaining the software is no longer sustainable or desirable (because it is no longer profitable, in a given sense). For example, accepting to lose users rather than adding colour-blind-friendly features to a software.

Another case is when all Players (or those with higher power over the software/play) decide to forcefully end the play. For example, new Rules are enforced by the environment (e.g., a change in an operative system), and the management (Players) decide to change Goals by no longer providing portability for that operating system, thus forcing that particular user-group (Players) to stop playing.

3.3. Rules

The Rules of an infinite game will determine what the Players are allowed to do in order to "win". However, given that there is no finite winning condition on an infinite game, the Rules will allow to continue playing.

Applied to TDM, this means that the stakeholder groups defined as Players will enforce rules upon each other—they are seldom agreed but often enforced given the power dynamics (Cico, 2019). However, research about what drives those decisions is scarce (Nielsen et al., 2020; Malakuti and Ostroumov, 2020), and most solutions focus on "engineering measures and feeding them into an idealised decision-making process" (Becker et al., 2018), rather than adapting them as the software (and thus the "game" of its sustainability) continues to evolve.

Moreover, TD itself can change and evolve (Li et al., 2015). Although TD management is a cross-cutting process abstracted beyond the TD types, the taxonomies of TD types upon which it is based evolve over time (Griffith et al., 2014; Freire et al., 2020); for example, "Algorithm Debt" (Liu et al., 2020) was not part of the seminal TD taxonomy (Alves et al., 2016). Paradoxically, deciding which TD type to work on is another Goal that the Players can agree (or disagree) on.

However, the TD types that are considered at a given moment during a play will provide applied rules for the game. This means that the techniques for measurement, identification, repayment, interest calculation, and others, can vary per type but are only a Rule if mitigating/discovering that TD type is also a Goal. *Note:* Discussing TD taxonomies and specific measurements or tools is out of scope of this position paper, as we focus on the overarching concept of framing TDM as an infinite game, rather than focusing on specific concepts.

3.4. Goals

A finite game is played to be won (this is the Goal), while an infinite one is played to outwear the other Players (Carse, 2011). As a result, the outcome of a play is an infinite path that could be simplified as the system's execution (Berwanger and Mathew, 2017).

[2] https://www.theserverside.com/feature/Why-GitHub-renamed-its-master-branch-to-main.

[3] https://nira.com/slack-history/.

When we discussed PLAYERS (see Section 3.2), we did so in terms of *stakeholder groups*; this is a common approach used in other areas of software development (Hujainah et al., 2018). However, each group is composed of *individual stakeholders*, which will vary in each game (software lifecycle), and even through time. Therefore, in *infinite TDM*, it is possible to have:

- G1. A set of "common goal" defined or agreed upon by a group. For example, all developers agree to favour quality over number of features, or all the users share an implicit interest because they are part of the same user market.
- G2. Individual goals inside a particular stakeholder group. These are defined purely by the individual person, without considering what the group goals are.

The specific goals of the system cannot be determined because they are given by the PLAYERS, for the context of a GAME (and each game is unique (Holtmann et al., 2010)). Therefore, (a) there can be an infinite number of goals and (b) those goals will continue to evolve as the game ensues (Berwanger and Doyen, 2020). Understanding stakeholders changeable desires has been (and continues to be) fruitful, extensive areas of research (Hujainah et al., 2018; Evertse et al., 2021; Mutabaruka, 2021; Griffith et al., 2014) that cannot be summarised in the position paper, nor they should be. The goal of this position is to outline the generic cases, but each GAME must be handled individually.

Related to this, assuming a group goal will remain stable throughout the infinite game, will eventually hamper the continuation of the game by limiting the adaptability. For example, in the Slack example, if the developers/management had ignored the user preferences and continued with video-game development, the company might have been unsuccessful (thus, ending the game).

Goal Conflicts. Given the extreme variability of GOALS (because of the amount of PLAYERS and the TIME/length of the infinite game), it is only natural that those goals will conflict with each other. Though the specific instances of conflicts cannot be outlined at the level this position works, it is possible to define four *types of conflicts* that may arise:

- C1. The common goals of two (or more) groups conflict with each other. For example, managers strive to expedite work at the cost of introducing TD, but the developers favour producing less features at a higher quality. It has been demonstrated that "software teams willing to accumulate TD could speed up functionality deployment in their products by as much as three times" (Ramasubbu et al., 2015).
- C2. An individual's goal conflicts with the goal of their own group. For example, an individual developer does not follow the practices mandated by their group, and releases/deploys untested code.
- C3. Individual goals conflicts with the goals of other groups.
- C4. Individual goals conflicts with other stakeholders' individual goals, whether within the same group or outside the group. For example, two developers having different programming practices disagree on how to approach a particular situation (namely, a "time marker").

These conflicts invariably results in trade-offs of goals, that will affect how TD is managed (Cico, 2019), and will have repercussions on the TDM process (Lenarduzzi et al., 2021). Software evolves, and new programming languages, life cycles, methodologies, and tools emerge with it, affecting its sustainability and thus the goals. (Venters et al., 2018).

Because an infinite player is aware of the ever-changing context of the game (and its four elements), an infinite approach should be prepared to be constantly transformed (Weber, 2020). As a result, the only shared goal is for all PLAYERS to continue profit from a software, regardless of what "profit" means for each of the involved stakeholder (Taivalsaari and Mikkonen, 2017). Likewise, this affects the RULES, since "the rules of an infinite game must change when players agree that the game is troubled by a finite outcome" (Holtmann et al., 2010). In this case, "finite outcomes" are immediate goals; for example, a specific TD detection, a new interest, a new way of using the software.

Therefore, upon a conflict, new GOALS have to be agreed. However, this agreement can:

(1) Be temporary.
(2) Generate a trade-off with other goals (namely, a goal is sacrificed to achieve this new one). For example, developers move a feature to another sprint in order to maintain the quality of the system.
(3) Become a new rule in the game (namely, de-facto standards).
(4) Replace a previous agreement. For example, following from (2), in the next sprint, speed and quantity of features is prioritised instead of quality.
(5) Lead to new TD, thus going against the idea of software sustainability (Venters et al., 2018).

Different frameworks have been proposed for managing this trade-off, as there is no universal solution—it always depends on the PLAYERS of a particular GAME, and their GOALS at a particular moment under a specific set of RULES. However, prior research determined that such trade-off balancing must consider (Ramasubbu et al., 2015): the probability of technology disruption, the customer satisfaction needs, and the reliability demanded by the business. In turn, these three points are the representation of what makes software "profitable" for each PLAYER.

Ideal Situation. Ideally, in order to accompany the *infinite game*, and to ensure PLAYERS continue to profit on the software, developers should ensure the software is stable, secure, fault-tolerant, and robust while doing so at a low cost with low effort (Rehman et al., 2018; Venters et al., 2018; Saputri and Lee, 2021), or the managers will no longer support the project, or the users would stop using the software unless enforced (Evertse et al., 2021; Kula and Robles, 2019). Nevertheless, because software systems involve some degree of human–computer interaction (Calp and Akcayol, 2015), it is not possible to prescribe this situation for all games, and is instead considered as the *ideal*.

4. Reframing TDM activities

Existing research in TDM identified eight key activities: identification, monitoring, communication, documentation, measurement, prioritisation, prevention, and repayment (Li et al., 2015). These fit in project management phases, and sometimes may belong to more than one (Ramasubbu and Kemerer, 2018): (1) DISCOVERY (includes identification, monitoring, communication and documentation), (2) COST–BENEFIT ANALYSIS (includes measurement, monitoring and prioritisation), and (3) CONTROLLING (implying prevention and repayment).

The proposal of infinite TDM does not alter *what* the activities do nor their overall purpose. However, it extends as to *why* they are relevant, and thus *how* they are conducted by linking them to the four concepts described in Section 3. Because these activities are completed by the PLAYERS, they "should be able to maintain a constant pace indefinitely" as considered in agile development (Mutabaruka, 2021); namely, the activities should be organised in a way that does not strain players (i.e., producing undue effort) nor hinder regular development activities (i.e., if

they imply too much additional work, these activities will be avoided, thus possibly incurring more TD). Moreover, they will have to vary constantly as the GOALS morph while the game advances.

The new relationship between the four concepts and the activities/process segments is summarised in Fig. 1 and discussed below. Note that in the sections below, the numbers match those in the Figure, and the internal activities are highlighted in bold; for example, "Identify TD" and "Monitor, Document and Communicate" are part of Discovery TD, which is Phase 1, hence number "1" for both in the picture.

4.1. TD discovery

Quality assurance and monitoring practices can link defects and decisions to known problems, acknowledge TD, and facilitate the quantification of debt-related interest (Ramasubbu and Kemerer, 2018). These activities are *done* by some players *to assist* other players *perform* CONTROLLING steps, and *pass* and *assess* the software at different milestones. There are six previously identified activities (Li et al., 2015) that fit in DISCOVERY. Note that in Fig. 1, these are depicted by the activities—identification, monitoring, documentation, communication, controlling, and cost–benefit analysis.

4.1.1. Identification

Identification is the detection of TD with specific techniques, which can lead to assess a TIME marker, which in turn can generate to a change in GOALS. This activity is essential to understand what TD type is being dealt within in the software, where the TD instances are, and what can/should be done about them. Therefore, identifying each instance:

A. Depends on the TD type that is being sought (Lenarduzzi et al., 2021). Prior studies have uncovered which techniques can be used for each particular TD type (Alfayez et al., 2020). For example, *code and dependency analysis* can be automated to identify violations of coding rules, lack of tests, and other issues.
B. Deciding to use a given type of identification technique is an *agreement* that should be discussed between PLAYERS. It can affect GOALS, and cause trade-offs, both between TD types or between goals (see Section 3.4). The first trade-off kind exists because not all TD types (or instances) can be minimised equally (Alves et al., 2016). The second exists because deciding to scan the software for specific TD types or instances is a GOAL in itself, and can thus conflict with other PLAYERS' goals.
C. The techniques used for TD identification may become a RULE during a period in the *infinite game*. For example, internal processes may change in order to use an identification technique.

Example: Lack of TDM, more specifically, TD identification, can allow instances to be undetected until it is too late. An example is a recent vulnerability in Log4j on December 2021, which was discovered only after more than 840k cyberattacks were detected within two days of the discovery.[4] In this case, after fixing another issue, developers unknowingly introduced this vulnerability. This is and example of Security Debt (Rindell et al., 2019).

Proposal: This type of analysis needs to be part of the release cycle, and ensure that TD occurrences are always visible. Though this, TD identification will become part of the DevOps cycle (Liu et al., 2020). However, TD identification must include a trade-off analysis between the goals. This should be clear, expedite, and will be part of the play history (namely, it should be traceable); having these characteristics will allow leading the *infinite TDM* towards a semi-perfect information game that then could be decided. Moreover, if TD is not identified and is allowed to proliferate in the software, then it will hamper the shared goal of "profit" from the software.

4.1.2. Monitoring & communication

This pair enables stakeholders to be aware of unresolved TD and decide what to do about it. These are among the most time-consuming and least effective steps (Liu et al., 2020), as TD occurrences are associated with a lack of progress and waste of time, having a negative influence on developers' morale (Besker et al., 2020). Moreover, developers tend to report the same TD, requiring a manual component in the identification and monitoring processes (Zazworka et al., 2013).

However, these two activities also determine the RULES of the game, meaning, what the players should do to continue playing; for example, repay a particular TD occurrence. Thus, the specific process used for monitoring and communicating may (i) change while the software exist, (ii) depend on specific TD types, and (iii) be replaced or removed suddenly.

[4] Reported by TechSpot at: https://www.techspot.com/news/92633-hackers-launch-over-840000-attacks-through-log4j-flaw.html, and OpenTex: https://blogs.opentext.com/log4j-vulnerability-explained-and-how-to-respond/.

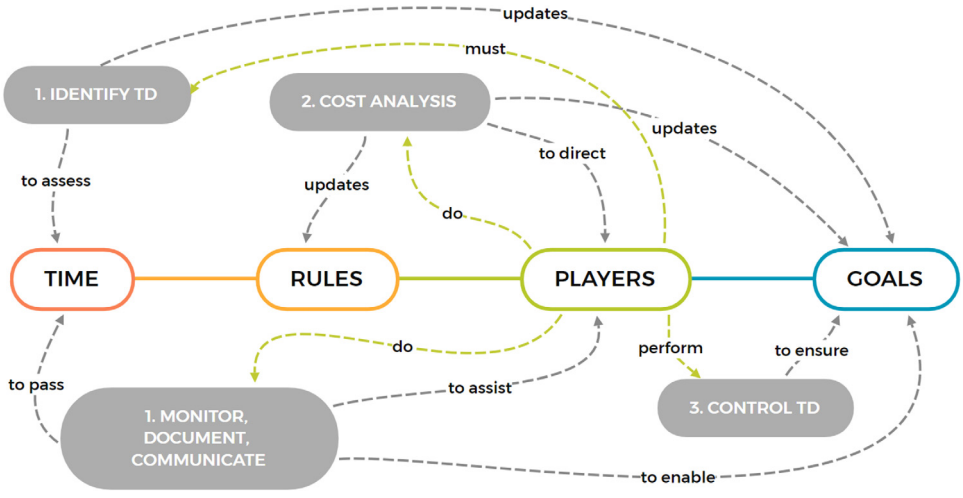

Fig. 1. TDM activities related to Infinite TDM concepts, grouped per process segments.

Example: Any piece of software inherently has bugs, regardless of the effort put in the development (Zhou, 2018); the same could be said for TD. Using the lean process (Meier and Liker, 2005), TD could be defined as a *problem* rather than a *bug*; this means defining TD in terms of the current and desired performance, the magnitude of the problem, and the extent and characteristics of the problem. This approach is currently being used in the industry for software bug detection,[5] based on the idea that vulnerabilities (i.e., TD in this case) have to be measured to 'exist'.[6]

Proposal: (a) These activities must be fully incorporated into the development process to ensure that they are performed in parallel (namely, akin to what test-driven development does with the inclusion of tests), (b) it cannot be assumed that a pre-assessed piece of code is free of TD, and it must instead be continuously monitored, (c) the monitoring must extend beyond the code to include stakeholders' as an additional source of debt (e.g., through unintended uses that become normative, or environmental/contextual changes).

4.1.3. Documentation

Documentation pertains to representing and codifying TD in an uniform manner. Though there are multiple approaches, companies tend not to follow a reporting template, and the approach used between companies changes, hindering developers' mobility and performance (Holvitie and Leppänen, 2013). Prior research has shown that TD documentation is seen as a waste of time both for the development team and for other stakeholders if not properly organised (Alahyari et al., 2019) ("waste" is used in a Lean context for a Lean/Agile software development process). This is problematic, as documentation allows the "play history" to be shared across the Players, which in turn will allow those players to establish their own Goals (either as individuals or groups), or even goals for a particular Time marker.

Moreover, as with other activities, documenting is a Rule in itself, as it will constrict some Players (e.g., developers or managers) to perform certain activities, while providing information to others (e.g., managers). The latter could also lead to a change in Goals.

Proposal: TD should be tackled as a requirement in itself and as a new feature. Processes to document TD (regardless of its type) tend to be static and predefined for an entire project (Lenarduzzi et al., 2021; Alfayez et al., 2020). However, this proposal argues that, because documentation processes are Rules, they should be flexible to adapt, akin as to how processes are perceived in agile methodologies (Mutabaruka, 2021). Likewise, incorporating all Players in the process of TD documentation could be beneficial in the long-term, since it would allow tackling the "finite outcomes" that occasionally trouble the infinite game. *Example:* Aligned with this, currently some companies have begun recruiting "technical writers" that harmonise developer-written documentation (Angelini, 2018); something similar could be investigated for TD.

4.2. Cost–benefit analysis

TD cannot be resolved at once or entirely eliminated since there will always be trade-offs. Thus, a team needs to quantify TD (namely, *do* a cost–benefit analysis) *to direct* the players by providing realistic, actionable policies related to TDM (Ramasubbu and Kemerer, 2018). These analyses will also *update* the Rules (e.g., by determining if a process step is worth taking) and thus allow assessing the "game" (namely, the play stated) on the next Time marker.

Infinite TDM is about the sustainability of software in the long term (namely, to keep playing and profit from the software). Therefore, the cost-benefit relationship of TD can lead to the Goals being updated, thus bringing further trade-offs (as discussed in Section 3.4). Moreover, the specific, TD-type-dependent processes used to perform this analysis are Rules in itself as well.

The activities that characterise this step are **measurement** (estimating the overall TD and quantifying the benefit and cost of each TD instance), **monitoring** (watching the changes in the cost/benefit of existing, unsolved debt), and **prioritisation** (ranking discovered debt according to rules, supporting decision-making, and solution strategies) (Li et al., 2015).

Example: A key element to recognise is that trade-offs can have impacts in the short-term and long-term. Sometimes, the goals for the same stakeholder can conflict. For example, a manager may want to deliver a high-quality product to avoid users' backlash, but may be compelled to deploy immediately (at the cost of lower quality) to start reaping the benefits to sustain the continued development. Likewise, the opposite is also a conflict between short-term and long-term goals. For example, the game "Halo Infinite" obtained early negative feedback (excessive bugs, poor interface, among others), the managers prioritised the long-term goals of a steady user base (given the multiplayer component), and thus delayed the release to improve the game, rather than favouring short-term goal of sales.[7]

Proposal: Current research focuses on adapting a rigid TDM process into a company's software development lifecycle, rather than providing them with the flexibility to adapt such process (Martini et al., 2018; Buchmann and Haki, 2021; Besker et al., 2018). Because some Players (i.e., users, environment, other related software systems) change, the measurements, tools and processes used in this step cannot remain static. If this does not happen, players are forced to incur additional costs and effort to alter their TDM; this may be even considered Process Debt (Alves et al., 2016). However, updating the cost–benefit analysis must include updating measures, their reference values, how it is monitored, and which conditions lead to a categorisation of priority.

4.3. Controlling TD

Players *perform* specific actions to control TD and *ensure* the Goals of software sustainability are met. Overall, this means for each Player (or stakeholder group) to continue profit from the software. In terms of TD, ensuring sustainability implies satisfying the maintenance obligations arising from shortcuts taken during the design, development, and deployment of software systems (Alves et al., 2016).

Like on prior steps, every activity is a Rule in itself. Controlling also allows assessing which Goals can be achieved. It is possible for an infinite game to be troubled by a (temporal) finite outcome (Holtmann et al., 2010) that act as the Time markers in which a system is assessed. As a result, it is possible that this activity is used as the source of information needed to reach an agreement for the conflicting Goals (see "Goal Conflicts" in Section 3.4) (Ramasubbu et al., 2015).

This step is conducted through **prevention** (blocking potential TDs from being incurred) and **repayment** (resolving or mitigating TD with techniques such as re-engineering and refactoring) (Codabux et al., 2014). As in prior activities, the specific processes involved in each activity will depend on the game itself (namely the software) and will vary per TD-type (Lenarduzzi et al., 2021; Alfayez et al., 2020; Li et al., 2015).

[5] See note: https://bit.ly/3sxQA2f.

[6] Derived from Brené Brown's studies in human connection: https://bit.ly/3onL3Ke.

[7] Read more: https://bit.ly/3AUwOS9.

Example: 'Business continuity planning' (BCP) (Srivastava, 2020) is essential for crisis management and to respond to disruptive events (e.g., cyber attacks, geopolitical turmoil, etc.). Even the smallest occurrence of TD can have a long-term effect that could potentially (in an unknown future timeframe) lead to a catastrophic failure due to its repercussions. A specific example is discussed in Section 5.

Proposal: In game theory, it is possible to play a succession of finite games within an infinite game (Weber, 2020; Holtmann et al., 2010). This means that TDM is infinite when looked at from a managerial perspective. Every TD instance (or occurrence) can be played as a self-contained finite game (i.e., incur, identify, document, communicate, repay, prevent). As a result, the consequences of a finite game (namely, the lifecycle of a particular TD occurrence) can be infinite (i.e., affecting the entire software for the remainder of its existence). For example, a TD occurrence can be easily solved, but the damage, financial risk and other liabilities it caused (e.g., users' mistrust leading to low usability) can continue in the long run, even after that TD repayment (Codabux et al., 2017).

This can directly affect the players, making some of them withdraw from the "game" (i.e., by deciding the software is no longer helpful and there is no need to keep using it). For example, if a specific TD repayment delays a system update, it may reduce the user base, effectively hindering the goal of keeping the software functional, relevant, robust, and stable. Though the repayment "concludes" the "finite game" of that occurrence, regaining the software's users' trust is the "infinite consequence" of the original TD.

5. Research avenues

Given the early stage of the proposal, we put forward several *research goals* to direct future efforts for infinite TDM. These are aligned to the Infinite Management Activities of Section 4. Each subsection includes a brief description of possible avenues, and lists thought-provoking rhetorical questions meant to spark future investigations. Additionally, each subsection includes one ongoing illustrative examples aimed at lowering the abstraction of our proposed avenues of work.

5.1. Avenues: Discovering TD

Having an automated process that performs the Discovery activities automatically is crucial for a change of mindset towards infinite TD. It would reduce the effort of constantly perusing the software project while maintaining it; however, such a process cannot be fully automated, so incorporating a manual efficient and effective component is essential. Considering TD as another software requirement can change the developers' "interest" in repaying TD; for example, this could be supported through TD visualisations adapted from burndown charts or dashboards, to be discussed in Scrum meetings. Recent investigations suggest that TDM varies depending on a company's maturity, sometimes dedicating excessive time to *prevention* rather than streamlining the process (Apa et al., 2020). As a result, relying only on data from large-scale companies would only provide partial insights; thus, such topics should also be adjusted for the type and maturity of the companies being assessed.

As a result, the following rhetorical questions could further direct TDM research towards an infinite framework: How can TD discovery and assessment be automatically added to the development lifecycle? Is a "DebtOps" process feasible? How to incorporate a manual assessment of incipient TD effectively and efficiently to complement the automated process? What types of visual reports promote an easier understanding of the existing TD, supporting the decision-making processes? What is the most effective and compelling way for developers to address TD as requirements?

Example. To lower the abstraction of these questions, we elaborate on the idea of a "DebtOps" process—a set of practices aiming to deploy and maintain TD discovery in production reliably and efficiently. Similar to MLOps: https://ml-ops.org.

It is worth considering that DevOps is a process with a high setup cost that lowers the effort during the lifecycle (Rubasinghe et al., 2018); however, DevOps is roughly "stable" unless one of the tools implementing it undergoes a considerable change. The main difference with the proposed (infinite) "DebtOps" is that DebtOps should be *flexible*, because the steps needed (or the importance given to a particular TD type or TD occurrence) can change on each iteration, depending on the business goals or environmental changes. Because the Rules change, some occurrences or TD types flagged as an issue before may no longer be an issue after a particular decision; likewise, such changes in the Rules can affect the interpretation of measurements. In the aforementioned Slack example, the entire DebtOps process would have required a complete rework.

Therefore, DebtOps research (and practice) could focus on understanding: (i) How to create a process/framework that is flexible but reliable and that modifying it requires limited effort, (ii) Human-centred aspects of the challenges faced by DebtOps developers, (iii) The cost and training issues of adapting to DebtOps, (iv) How reliable and traceable this process can be if measures and interpretations are changed as players continue playing the game (i.e., sustaining the software)

5.2. Avenues: Cost-benefit analysis

The entire TDM process must become agile and flexible enough to adapt to the ever-changing rules of infinite TD, and its imperfect information. This means that the measurements and indicators, must constantly adapt as the lifecycle progresses; e.g., values that were previously considered as acceptable, may not continue to be so in the future. Likewise, the current analysis structure must also be extended to evaluate players as TD generators, including measuring their effects and impact.

The following rhetorical questions could further direct TDM research into an infinite framework. How can business values be incorporated into the calculation of prioritisation? Does the recalculation and adjustment of cost–benefit analysis increase the agility of the process? How to incorporate "players" as part of the measurements used in the TD cost–benefit analysis? How to measure the long-lasting impact of specific TD smells from a financial perspective?

Example. Continuing with the "DebtOps" proposal of Section 5.1, we consider the incorporation of players as part of the measurements used in the TDM cost–benefit analysis.

Recently, Machine Learning (ML) algorithms have become popular and are being used in the data analysis in multiple domains and disciplines, especially in management (Savenkov and Ivutin, 2019). Therefore, one way to answer this research question would be to assess the feasibility of ML approaches to predict how players (i.e., stakeholders) will react to a change in the future based on their past attitude or actions. Then, this information could be incorporated in the cost-analysis benefit process. Moreover, such information may not be restricted only to a company or developers' previous projects, but over time with increased research and practice, pre-trained models (Qiu et al., 2020) could be developed for different software domains

(e.g., models for people downloading an Android app for to-do lists may not behave the same as those involved in a high-end console videogame project).

Such an approach should be part of the DebtOps framework since it could link back to the measures considered. However, it should be noted that not all stakeholders need to be included in the process at every single step of the way.

5.3. Controlling TD

New metrics, analyses and prioritisation should be created to evaluate the infinite consequences of the TD finite games (i.e., the lifecycle of a particular TD occurrence) over time. TD occurrences may have "hidden" effects on some players that undermine the ability of other players to sustain the goals in the infinite TD process (e.g., why maintain the software if the users no longer trust it or use it?). As a result, new research should focus on preventing and repaying those hidden effects.

The following rhetorical questions could further direct TDM research into an infinite framework. How to measure the long-lasting consequences of TD? What makes up the "unknown" costs of the long-term impact of TD? How does this translate to measurable variability? How to handle the unknown costs (and the debt's remnant effects) when TD resurfaces after being "dormant" for a period? Following the prevention and repayment of TD, how to handle its consequences on other players (e.g., lack of trust or interest in the software)?

Example. Continuing with the "DebtOps" example from Section 5.1, let us think about what constitutes the "unknown" costs of the long-term impact of TD.

Any repayment of TD will force developers to change the code, design or architecture. The problems arising because of those changes are not "isolated changes" but should be traced back to the original TD repayment; it has been demonstrated that traceability has a critical role in software sustainability (Tian et al., 2021). Therefore, companies can perform a retrospective analysis to investigate the impact of those changes on the original TD. As a result, this will mean "tracing" debt and changes. If the "original" debt had not been incurred, the developers would not have needed to repay it, and a whole spectrum of issues would not have spread into the system. By linking those "transitive" occurrences of debt back to the original one, players will see the "real" cost of the debt (and everything that it implied), possibly confirming that a debt occurrence is infinite.

6. Conclusions

TDM is an inherent part of every software development lifecycle. Each TD occurrence can have infinite consequences, affecting the ultimate goal of sustaining the software over time to keep it relevant, useful and stable. TD can also affect the players and, through them, incur on further TD instances that affect the goals sustainability, effectively making this an infinite cycle that will continue as long as the software remains usable.

This paper proposed a shift of mindset regarding how TDM is approached and embedded in the software lifecycle. Without using mathematical-centric proposals, we related TD management to infinite games in game theory, defining it in terms of players, goals, rules (or lack thereof) and time markers. Then, we discussed how these four components affect the activities and processes part of TDM. Overall, this position paper calls for the community to test and apply the proposed ideas in practice, validate them, refine them, or even oppose them. It aims to stimulate reflective thinking and promote a novel and different line of action and research among software engineers and researchers present and future.

CRediT authorship contribution statement

Melina Vidoni: Conceptualisation of this study, Methodology, Writing. **Zadia Codabux:** Methodology, Writing. **Fatemeh H. Fard:** Methodology, Writing.

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.

Acknowledgements

This study is partly supported by the Natural Sciences and Engineering Research Council of Canada, RGPIN-2021-04232 and DGECR-2021-00283 at the University of Saskatchewan, and RGPIN-2019-05175 at the University of British Columbia.

References

Alahyari, H., Gorschek, T., Berntsson Svensson, R., 2019. An exploratory study of waste in software development organizations using agile or lean approaches: A multiple case study at 14 organizations. Inf. Softw. Technol. 105, 78–94. http://dx.doi.org/10.1016/j.infsof.2018.08.006.

Alfayez, R., Alwehaibi, W., Winn, R., Venson, E., Boehm, B., 2020. A systematic literature review of technical debt prioritization. In: 3rd International Conference on Technical Debt. In: TechDebt '20, Association for Computing Machinery, USA, pp. 1–10. http://dx.doi.org/10.1145/3387906.3388630.

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. http://dx.doi.org/10.1016/j.infsof.2015.10.008.

Angelini, G., 2018. Current practices in web API documentation. In: Proceedings of the European Academic Colloquium on Technical Communication. pp. 70–85.

Apa, C., Jeronimo, H., Nascimento, L.M., Vallespir, D., Travassos, G.H., 2020. The perception and management of technical debt in software startups. In: Fundamentals of Software Startups: Essential Engineering and Business Aspects. Springer International Publishing, Cham, pp. 61–78. http://dx.doi.org/10.1007/978-3-030-35983-6_4.

Axelsson, J., 2019. Game theory applications in systems-of-systems engineering: A literature review and synthesis. Procedia Comput. Sci. 153, 154–165. http://dx.doi.org/10.1016/j.procs.2019.05.066, 17th Annual Conference on Systems Engineering Research (CSER).

Bavota, G., Oliveto, R., De Lucia, A., Antoniol, G., Guéhéneuc, Y.-G., 2010. Playing with refactoring: Identifying extract class opportunities through game theory. In: 2010 IEEE International Conference on Software Maintenance. pp. 1–5. http://dx.doi.org/10.1109/ICSM.2010.5609739.

Becker, C., Chitchyan, R., Betz, S., McCord, C., 2018. Trade-off decisions across time in technical debt management: A systematic literature review. In: 2018 International Conference on Technical Debt. In: TechDebt '18, ACM, NY, USA, pp. 85–94. http://dx.doi.org/10.1145/3194164.3194171.

Berwanger, D., Doyen, L., 2020. Observation and distinction. Representing information in infinite games. In: Paul, C., Bläser, M. (Eds.), 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020). In: Leibniz International Proceedings in Informatics (LIPIcs), 154, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 48:1–48:17. http://dx.doi.org/10.4230/LIPIcs.STACS.2020.48.

Berwanger, D., Mathew, A.B., 2017. Infinite games with finite knowledge gaps. Inform. and Comput. 254, 217–237. http://dx.doi.org/10.1016/j.ic.2016.10.009, SR 2014.

Besker, T., Ghanbari, H., Martini, A., Bosch, J., 2020. The influence of technical debt on software developer morale. J. Syst. Softw. 167, 110586. http://dx.doi.org/10.1016/j.jss.2020.110586.

Besker, T., Martini, A., Edirisooriya Lokuge, R., Blincoe, K., Bosch, J., 2018. Embracing technical debt, from a startup company perspective. In: 2018 IEEE International Conference on Software Maintenance and Evolution. ICSME, pp. 415–425. http://dx.doi.org/10.1109/ICSME.2018.00051.

Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., Engler, D., 2010. A few billion lines of code later: Using static analysis to find bugs in the real world. Commun. ACM 53 (2), 66–75. http://dx.doi.org/10.1145/1646353.1646374.

Buchmann, L., Haki, K., 2021. Digital nudging for technical debt management: Insights from a technology-driven organization. In: Hawaii International Conference on System Sciences. HICSS 54, pp. 1–12, URL https://www.alexandria.unisg.ch/261573/.

Calp, M.H., Akcayol, M.A., 2015. The importance of human-computer interaction in the development process of software projects. Glob. J. Inf. Technol. Emerg. Technol. 5 (1), 48–54.

Carse, J., 2011. Finite and Infinite Games. Simon and Schuster.

Cico, O., 2019. Technical debt trade-off - experiences from software startups becoming grownups. In: Software Business. pp. 413–418. http://dx.doi.org/10.1007/978-3-030-33742-1.

Codabux, Z., Williams, B.J., Bradshaw, G.L., Cantor, M., 2017. An empirical assessment of technical debt practices in industry. J. Softw. Evol. Process 29 (10), e1894.

Codabux, Z., Williams, B.J., Niu, N., 2014. A quality assurance approach to technical debt. In: Proceedings of the International Conference on Software Engineering Research and Practice. SERP, The Steering Committee of The World Congress in Computer Science, pp. 1–7.

Cui, G., Li, M., Wang, Z., Ren, J., Jiao, D., Ma, J., 2015. Analysis and evaluation of incentive mechanisms in P2P networks: a spatial evolutionary game theory perspective. Concurr. Comput.: Pract. Exper. 27 (12), 3044–3064. http://dx.doi.org/10.1002/cpe.3207.

D'Alotto, L., 2020. Infinite games on finite graphs using grossone. Soft Comput. 24 (23), 17509–17515. http://dx.doi.org/10.1007/s00500-020-05167-1.

Dziembowski, S., Jurdzinski, M., Walukiewicz, I., 1997. How much memory is needed to win infinite games? In: Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science. pp. 99–110. http://dx.doi.org/10.1109/LICS.1997.614939.

Evertse, R., Lencz, A., Šinik, T., Jansen, S., Soussi, L., 2021. Is your software ecosystem in danger? Preventing ecosystem death through lessons in ecosystem health. In: Gregory, P., Kruchten, P. (Eds.), Agile Processes in Software Engineering and Extreme Programming – Workshops. Springer International Publishing, Cham, pp. 96–105.

Fernando, D., 2018. Model checking Nash-equilibrium - automatic verification of robustness in distributed systems. In: Sun, J., Sun, M. (Eds.), Formal Methods and Software Engineering. Springer International Publishing, Cham, pp. 436–440.

Freire, S., Rios, N., Mendonça, M., Falessi, D., Seaman, C., Izurieta, C., Spínola, R.O., 2020. Actions and impediments for technical debt prevention: Results from a global family of industrial surveys. In: 35th Annual ACM Symposium on Applied Computing. Association for Computing Machinery, USA, pp. 1548—1555.

Garcia, J., Ivkovic, I., Medvidovic, N., 2013. A comparative analysis of software architecture recovery techniques. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 486–496. http://dx.doi.org/10.1109/ASE.2013.6693106.

Gavidia-Calderon, C., Sarro, F., Harman, M., Barr, E.T., 2020. Game-theoretic analysis of development practices: Challenges and opportunities. J. Syst. Softw. 159, 110424. http://dx.doi.org/10.1016/j.jss.2019.110424.

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. pp. 1014–1025. http://dx.doi.org/10.1109/WSC.2014.7019961.

Hesari, S., Mashayekhi, H., Ramsin, R., 2010. Towards a general framework for evaluating software development methodologies. In: 2010 IEEE 34th Annual Computer Software and Applications Conference. pp. 208–217. http://dx.doi.org/10.1109/COMPSAC.2010.69.

Holtmann, M., Kaiser, Ł., Thomas, W., 2010. Degrees of lookahead in regular infinite games. In: Ong, L. (Ed.), Foundations of Software Science and Computational Structures. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 252–266.

Holvitie, J., Leppänen, V., 2013. DebtFlag: Technical debt management with a development environment integrated tool. In: 2013 4th International Workshop on Managing Technical Debt. MTD, pp. 20–27. http://dx.doi.org/10.1109/MTD.2013.6608674.

Hu, Z., Wu, W., 2014. A game theoretic model of software crowdsourcing. In: 2014 IEEE 8th International Symposium on Service Oriented System Engineering. pp. 446–453. http://dx.doi.org/10.1109/SOSE.2014.79.

Hujainah, F., Bakar, R.B.A., Abdulgabber, M.A., Zamli, K.Z., 2018. Software requirements prioritisation: A systematic literature review on significance, stakeholders, techniques and challenges. IEEE Access 6, 71497–71523. http://dx.doi.org/10.1109/ACCESS.2018.2881755.

Khan, A.A., Abolhasan, M., Ni, W., 2018. An evolutionary game theoretic approach for stable and optimized clustering in VANETs. IEEE Trans. Veh. Technol. 67 (5), 4501–4513. http://dx.doi.org/10.1109/TVT.2018.2790391.

Kokol, P., Kokol, M., Zagoranski, S., 2021. Code smells: A synthetic narrative review. CoRR abs/2103.01088 arXiv:2103.01088.

Krishna, V., Basu, A., 2012. Minimizing technical debt: developer's viewpoint. IET Conf. Proc. (1), 14. http://dx.doi.org/10.1049/ic.2012.0147.

Kruchten, P., Nord, R.L., Ozkaya, I., Falessi, D., 2013. Technical debt: Towards a crisper definition report on the 4th international workshop on managing technical debt. SIGSOFT Softw. Eng. Notes 38 (5), 51–54. http://dx.doi.org/10.1145/2507288.2507326.

Kula, R.G., Robles, G., 2019. The life and death of software ecosystems. In: Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability: Communications of NII Shonan Meetings. Springer Singapore, Singapore, pp. 97–105. http://dx.doi.org/10.1007/978-981-13-7099-1_6.

Laybats, C., Davies, J., 2018. GDPR: Implementing the regulations. Bus. Inform. Rev. 35 (2), 81–83. http://dx.doi.org/10.1177/0266382118777808.

Le Roux, S., Pauly, A., 2020. A semi-potential for finite and infinite games in extensive form. Dynam. Games Appl. 10 (1), 120–144. http://dx.doi.org/10.1007/s13235-019-00301-7.

Lee, E., Seo, Y.-D., Kim, Y.-G., 2019. A Nash equilibrium based decision-making method for internet of things. J. Ambient Intell. Humaniz. Comput. http://dx.doi.org/10.1007/s12652-019-01367-2.

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.

Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw. 101, 193–220. http://dx.doi.org/10.1016/j.jss.2014.12.027.

Liu, J., Huang, Q., Xia, X., Shihab, E., Lo, D., Li, S., 2020. Is using deep learning frameworks free? Characterizing technical debt in deep learning frameworks. In: 42nd International Conference on Software Engineering: Software Engineering in Society. ACM, NY, USA, pp. 1–10. http://dx.doi.org/10.1145/3377815.3381377.

Malakuti, S., Ostroumov, S., 2020. The quest for introducing technical debt management in a large-scale industrial company. In: Jansen, A., Malavolta, I., Muccini, H., Ozkaya, I., Zimmermann, O. (Eds.), Software Architecture. Springer International Publishing, Cham, pp. 296–311.

Martini, A., Besker, T., Bosch, J., 2018. Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. Sci. Comput. Programm. 163, 42–61. http://dx.doi.org/10.1016/j.scico.2018.03.007.

Meier, D., Liker, J., 2005. The Toyota Way Fieldbook. McGraw-Hill, US, p. 1. http://dx.doi.org/10.1036/0071448934, URL https://mhebooklibrary.com/doi/book/10.1036/0071448934.

Mutabaruka, E., 2021. Agile methodology software development adaptability challenges in corporate organization. Available at SSRN 3851349.

Nielsen, M.E., Østergaard Madsen, C., Lungu, M.F., 2020. Technical debt management: A systematic literature review and research agenda for digital government. In: Viale Pereira, G., Janssen, M., Lee, H., Lindgren, I., Rodríguez Bolívar, M.P., Scholl, H.J., Zuiderwijk, A. (Eds.), Electronic Government. Springer International Publishing, Cham, pp. 121–137.

Ozkan-Canbolat, E., Beraha, A., Bas, A., 2016. Application of evolutionary game theory to strategic innovation. Procedia - Soc. Behav. Sci. 235, 685–693. http://dx.doi.org/10.1016/j.sbspro.2016.11.069, 12th International Strategic Management Conference, ISMC 2016, 28-30 October 2016, Antalya, Turkey.

Pujols, J.B., Bas, P., Martínez-Fernandez, S., Martini, A., Trendowicz, A., 2020. Skuld: A self-learning tool for impact-driven technical debt management. In: Proceedings of the 3rd International Conference on Technical Debt. In: TechDebt '20, Association for Computing Machinery, New York, NY, USA, pp. 113–114. http://dx.doi.org/10.1145/3387906.3388626.

Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., Huang, X., 2020. Pre-trained models for natural language processing: A survey. Sci. China Technol. Sci. 63 (10), 1872–1897. http://dx.doi.org/10.1007/s11431-020-1647-3.

Ramasubbu, N., Kemerer, C., 2018. Integrating technical debt management and software quality management processes: A framework and field tests. In: 40th International Conference on Software Engineering. In: ICSE '18, ACM, NY, USA, p. 883. http://dx.doi.org/10.1145/3180155.3182529.

Ramasubbu, N., Kemerer, C.F., 2019. Integrating technical debt management and software quality management processes: A normative framework and field tests. IEEE Trans. Softw. Eng. 45 (3), 285–300. http://dx.doi.org/10.1109/TSE.2017.2774832.

Ramasubbu, N., Kemerer, C.F., Woodard, C.J., 2015. Managing technical debt: Insights from recent empirical evidence. IEEE Softw. 32 (2), 22–25. http://dx.doi.org/10.1109/MS.2015.45.

Rehman, F.u., Maqbool, B., Riaz, M.Q., Qamar, U., Abbas, M., 2018. Scrum software maintenance model: Efficient software maintenance in agile methodology. In: 2018 21st Saudi Computer Society National Computer Conference. NCC, pp. 1–5. http://dx.doi.org/10.1109/NCG.2018.8593152.

Rindell, K., Bernsmed, K., Jaatun, M.G., 2019. Managing security in software: Or: How I learned to stop worrying and manage the security technical debt. In: Proceedings of the 14th International Conference on Availability, Reliability and Security. In: ARES '19, Association for Computing Machinery, USA, pp. 1–8. http://dx.doi.org/10.1145/3339252.3340338.

Rios, N., Freire, S., Pérez, B., Castellanos, C., Correal, D., Mendonca, M., Falessi, D., Izurieta, C., Seaman, C.B., Spinola, R.O., 2021. On the relationship between technical debt management and process models. IEEE Softw. http://dx.doi.org/10.1109/MS.2021.3058652.

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. http://dx.doi.org/10.1016/j.infsof.2018.05.010.

Rubasinghe, I., Meedeniya, D., Perera, I., 2018. Traceability management with impact analysis in DevOps based software development. In: International Conference on Advances in Computing, Communications and Informatics. ICACCI, pp. 1956–1962. http://dx.doi.org/10.1109/ICACCI.2018.8554399.

Sandholm, W.H., 2020. Evolutionary game theory. Complex Soc. Behav. Syst. Game Theory Agent-Based Models 573–608.

Saputri, T.R.D., Lee, S.-W., 2021. Integrated framework for incorporating sustainability design in software engineering life-cycle: An empirical study. Inf. Softw. Technol. 129, 106407. http://dx.doi.org/10.1016/j.infsof.2020.106407.

Savenkov, P.A., Ivutin, A.N., 2019. Methods and algorithms of data and machine learning usage in management decision making support systems. In: 8th Mediterranean Conference on Embedded Computing. MECO, pp. 1–4. http://dx.doi.org/10.1109/MECO.2019.8760191.

Schütz, J., Gómez, J.M., 2020. Towards collaborative technical debt management in systems of systems. In: Proceedings of the 3rd International Conference on Technical Debt. In: TechDebt '20, Association for Computing Machinery, NY, USA, pp. 87–91. http://dx.doi.org/10.1145/3387906.3388620.

Seaman, C., Guo, Y., 2011. Measuring and Monitoring Technical Debt. In: Adv. Comput., 82, Elsevier, pp. 25–46. http://dx.doi.org/10.1016/B978-0-12-385512-1.00002-5.

Skourletopoulos, G., Mavromoustakis, C.X., Mastorakis, G., Batalla, J.M., Sahalos, J.N., 2017a. An evaluation of cloud-based mobile services with limited capacity: a linear approach. Soft Comput. 21 (16), 4523–4530. http://dx.doi.org/10.1007/s00500-016-2083-4.

Skourletopoulos, G., Mavromoustakis, C.X., Mastorakis, G., Sahalos, J.N., Batalla, J.M., Dobre, C., 2017b. A game theoretic formulation of the technical debt management problem in cloud systems. In: 2017 14th International Conference on Telecommunications. ConTEL, pp. 7–12. http://dx.doi.org/10.23919/ConTEL.2017.8000012.

Sohrabi, M.K., Azgomi, H., 2019. Evolutionary game theory approach to materialized view selection in data warehouses. Knowl.-Based Syst. 163, 558–571. http://dx.doi.org/10.1016/j.knosys.2018.09.012.

Srivastava, A., 2020. Playing the Infinite Game. WorldQuadrant Perspectives, URL https://www.weareworldquant.com/en/thought-leadership/playing-the-infinite-game/.

Taivalsaari, A., Mikkonen, T., 2017. A roadmap to the programmable world: Software challenges in the IoT era. IEEE Softw. 34 (1), 72–80. http://dx.doi.org/10.1109/MS.2017.26.

Thomas, W., 2002. Infinite games and verification. In: Brinksma, E., Larsen, K.G. (Eds.), Computer Aided Verification. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 58–65.

Tian, F., Wang, T., Liang, P., Wang, C., Khan, A.A., Babar, M.A., 2021. The impact of traceability on software maintenance and evolution: A mapping study. J. Softw. Evol. Process 33 (10), e2374. http://dx.doi.org/10.1002/smr.2374.

Tong, Z., Su, X., Cen, L., Wang, T., 2017. Greening software requirements change management strategy based on Nash equilibrium. Wirel. Commun. Mob. Comput. 2017, 4020162. http://dx.doi.org/10.1155/2017/4020162.

Tosh, D., Sengupta, S., Kamhoua, C., Kwiat, K., Martin, A., 2015. An evolutionary game-theoretic framework for cyber-threat information sharing. In: 2015 IEEE International Conference on Communications. ICC, pp. 7341–7346. http://dx.doi.org/10.1109/ICC.2015.7249499.

Venters, C.C., Capilla, R., Betz, S., Penzenstadler, B., Crick, T., Crouch, S., Nakagawa, E.Y., Becker, C., Carrillo, C., 2018. Software sustainability: Research and practice from a software architecture viewpoint. J. Syst. Softw. 138, 174–188. http://dx.doi.org/10.1016/j.jss.2017.12.026.

Venters, C.C., Jay, C., Lau, L., Griffiths, M.K., Holmes, V., Ward, R.R., Austin, J., Dibsdale, C.E., Xu, J., 2014. Software sustainability: The modern tower of babel. In: CEUR Workshop Proceedings, vol. 1216. CEUR, pp. 7–12.

Weber, N., 2020. Finite and infinite games: An ethnography of institutional logics in research software sustainability. Proc. Assoc. Inform. Sci. Technol. 57 (1), e281. http://dx.doi.org/10.1002/pra2.281.

Zazworka, N., Spínola, R.O., Vetro', A., Shull, F., Seaman, C., 2013. A case study on effectively identifying technical debt. In: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering. In: EASE '13, Association for Computing Machinery, USA, pp. 42–47. http://dx.doi.org/10.1145/2460999.2461005.

Zhou, C., 2018. Intelligent bug fixing with software bug knowledge graph. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. In: ESEC/FSE 2018, ACM, USA, pp. 944–947. http://dx.doi.org/10.1145/3236024.3275428.

Dr. Melina Vidoni is a newly joined academic (Lecturer, eq. to Assistant Professor) at the Australian National University in the CECS School of Computing, where she continues her domestic and international collaborations with Canada and Germany. She graduated from Universidad Tecnologica Nacional (UTN) as an Information Systems Engineer. Years later, she received her Ph.D. on the same institution, with the maximum qualification. She founded R-Ladies Santa Fe in 2018, and in 2019 moved to Australia to further her research career. Since 2018, she is also Associate Editor for rOpenSci, often acting as Editor in Chief. Dr Vidoni's main research interests are mining software repositories, technical debt, software development, and empirical software engineering when applied to data science and scientific software.

Dr. Zadia Codabux is an Assistant Professor in the Department of Computer Science at the University of Saskatchewan, Canada. Her research interests include technical debt, empirical software engineering, and software security. She completed her Ph.D. in Computer Science at Mississippi State University, USA and her MS in Computer Science at the University of Mauritius. She is a strong advocate for women in computing and is an attendee, speaker, and reviewer for the Computing Research Association's Committee on Widening Participation in Computing Research, Grace Hopper Celebration, and National Center for Women & Information Technology.

Dr. Fatemeh Hendijani Fard is an Assistant Professor at the University of British Columbia. With a background in anomaly detection of MultiAgent Systems, she is working on intelligent techniques and tools to help software engineers. Her recent research focuses on machine learning and natural language processing to analyse software artefacts including user feedback and source code. Dr. Fard has been the reviewer of multiple conferences and journals and has organised big data analysis workshops, including the Big Data Analytics, Industry and Academics at UBC Okanagan in 2019. She has worked on different industry-partnered projects and her group is working closely with companies on their research requirements. Dr. Fard is one of the instructors teaching data science courses in the Master of Data Science program, and she gives back to the community by mentoring women looking to get into a career in AI.