DESIGN METHODS FOR DIAGNOSING AND LOCATING ENTANGLED TECHNICAL DEBT IN DEVOPS FRAMEWORKS

Bonet Faus, José (1);
Le Masson, Pascal (1);
Pelissier, Ugo (1);
Jibet, Nafissa (1);
Bordas, Antoine (1);
Pajot, Sébastien (2)

1: Mines de Paris;
2: Ubisoft

ABSTRACT

In the IT landscape, DevOps is the preferred approach for developing and maintaining rapidly evolving systems that require continuous improvements. Yet, DevOps frameworks do not entirely prevent the accumulation of Technical Debt (TD), and under certain circumstances DevOps can even contribute to generating TD. This paper focuses on a specific type of TD, Entangled Technical Debt (ETD), that corresponds to the implicit complexification of a system's design and the appearance of unintentional couplings in its architecture over time. Our work seeks to inform methods for Diagnosing and Locating ETD in DevOps frameworks. Through a research partnership with Ubisoft's IT branch, an experimental case-study was conducted. It takes the form of an assessment of 6 innovative IT projects and a subsequent in-depth architecture analysis of an individual IT system, which enabled the characterization of the mechanisms linking DevOps to ETD. This allowed us to develop and test practical methods for diagnosing and locating ETD in IT systems.

Keywords: Technical Debt, DevOps, Design theory, Systems Engineering (SE), Innovation

Contact:
Bonet Faus, Jose
Ubisoft
France
pbonetfaus@gmail.com

Cite this article: Bonet Faus, J., Le Masson, P., Pelissier, U., Jibet, N., Bordas, A., Pajot, S. (2023) 'Design Methods for Diagnosing and Locating Entangled Technical Debt in DevOps Frameworks', in *Proceedings of the International Conference on Engineering Design (ICED23),* Bordeaux, France, 24-28 July 2023. DOI:10.1017/pds.2023.127

1 INTRODUCTION

1.1 IT design challenges

The adoption of digital technology by organizations to improve efficiency and innovation has become a mandatory practice across most industrial sectors. IT branches are at the helm of this transition, tasked with providing and maintaining quality working tools for all users. The systems they develop must operate in rapidly evolving environments and require frequent improvements: designing in IT is synonymous with designing over legacy infrastructure on which new features need to be continuously implemented. Thus, IT projects rarely start from scratch and frequently deal with the adaptation of old systems to ever-changing landscapes. This brings forth a particular set of design challenges, and the current standard to tackle these challenges is the adoption of iterative and incremental frameworks, such as DevOps.

1.2 DevOps frameworks

DevOps is a burgeoning field and as such an established academic definition is hard to come by. As a term, it comes from the combination of the words "Development" and "Operations". Zhu et al. (2016) suggested defining DevOps as **"a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality"**. To the best of our understanding DevOps is a method that combines organizational best practices and engineering best practices with the aim of improving the two main aspects of IT projects. On one hand, "development" through the use of Lean management (Ohno, 2019) and Agile Software Development (Beck et al., 2001) that aim at quickly responding to a user's need; therefore making DevOps a user-oriented approach. On the other hand, "operations" through the use of Continuous Integration / Continuous Delivery (Shahin et al., 2017) and Infrastructure as Code (Rahman et al., 2019) to address the fact that the developed features need to be incorporated into complex existing systems. However, some omnipresent factors in IT innovation such as continuous release frameworks that create strict deadlines and time pressure (Colomo-Palacios et al., 2020), can prevent an ideal implementation of DevOps practices. In the context of this paper, a system in a DevOps framework refers to a system that was originally developed with DevOps practices and continues to evolve in an environment that uses them.

1.3 Technical debt in DevOps frameworks

Technical Debt (TD) refers to **a collection of design choices and implementation constructs amassed during the development phase which set up a technical context that induces future costs**. It is a well known design challenge in IT, but the related academic body of work is still nascent.

1.4 Content overview

This paper strives to study this link and demonstrate that the adoption of a DevOps framework can contribute to an accumulation of a particular kind of TD known as Entangled Technical Debt (ETD). The contents of the paper were developed through a research partnership with Ubisoft's IT branch. In our brief literature review we will provide a definition of ETD and the reasoning behind its association with DevOps frameworks. Subsequently, we will state the Research Questions that we wish to address. Then we will explain our theoretical model and the associated methods developed for diagnosing and locating ETD in systems. Next, we will describe the empirical material in which said methods were deployed and present the results showing how they apply to the analysis of real use-cases in corporate IT systems. At last, we will discuss the paper's contributions, limits and future perspectives.

2 LITERATURE REVIEW

2.1 Defining technical debt

TD originated in the world of software development and is strongly anchored in it. Nonetheless, at its root, TD has all the makings of a design issue. TD comes from a metaphor inspired by financial debt (Cunningham, 1992). The debt metaphor only exists in order to foster awareness and encourage communication on said design problems. From a formal standpoint, TD refers to **a collection of design choices and implementation constructs amassed during the development phase which set up a**

https://doi.org/10.1017/pds.2023.127 Published online by Cambridge University Press

technical context that induces future costs. These costs can be interpreted as an interest rate. In IT, said rates take the form of a difficulty in making future changes, and they lead to adverse side-effects such as additional development time for refactoring or increasingly frequent bugs. In any case, the inevitable nature of the factors that contribute to TD accumulation leads us to believe that designing in IT is designing with TD.

2.2 Types of technical debt

The proposed definition for TD encompasses a broad spectrum of situations, and over the years, there has been an effort to establish TD classification systems. Li et al. (2015) published a comprehensive literature review and proposed a TD classification tree, consisting of 10 TD types. Fowler (2009) proposed a "Technical Debt Quadrant", in which he breaks down TD along two axes: reckless / prudent and deliberate / inadvertent. McConnell (2007) distinguishes unintentional and intentional debt, which in turn is broken up as tactical (reactive) versus strategic (proactive) debt. From these classification efforts, a series of individual TD types have been defined. In particular, two specific TD types raised our interest, since their characterizations resonated well with DevOps principles. Martini et al. (2014) give a detailed definition of **Architectural Technical Debt (ATD)** which refers to major design decisions (e.g., choices regarding structure, frameworks, technologies, languages, etc.) in software-intensive systems that, while being suitable or even optimal when made, entail a lack of changeability and evolvability that can significantly hinder progress in the future. Liodden (2020) defines **Bit Rot Technical Debt (BRTD)** as a debt that slowly happens over time. Components or systems devolve into unnecessary complexity through lots of incremental changes, resulting in tightly coupled architectures that require large parts of an organization to work as a unit in order to implement new features, significantly hindering changeability. The link between DevOps and BRTD is straight forward: DevOps frameworks favor a continuous succession of small incremental changes that can generate BRTD. The link between DevOps and ATD lies in the fact that DevOps frameworks are highly user-focused. They look to get closer to end-users and better understand their needs. In order to deliver on said needs, system functionalities need to be altered, and these functional changes can have profound architectural consequences. The adoption of a DevOps framework doesn't necessarily entail architectural instability, however there are no elements in DevOps that guarantee the integrity of non user-related design traits. Nothing warrants the perennity of a changeable and evolvable internal architecture, because it is not a user-related deliverable. For the sake of convenience, we have chosen to define a specific kind of Technical Debt that encompasses all elements of the aforementioned ATD and BRTD linked to DevOps. It has been named **Entangled Technical Debt (ETD)**, and it is defined as **an implicit complexification of a system's architecture and an unintentionally coupled design**, caused by a highly iterative and user-focused framework. This is the kind of TD that we will touch upon in the work that follows.

2.3 Need

In large scale DevOps frameworks, comprehending the mechanisms behind ETD accumulation is key to re-establishing IT systems as assets that create value for organizations. However, ETD is inherently difficult to grasp: diagnosing it and comprehending its effects is a challenging task. In order to provide structure, we propose to envision ETD as a collection of interdependencies between the functional needs and technical solutions of a system, which allows us to think of it as an architecture design issue. In the field of Design Science, theories such as Axiomatic Design (Suh and Suh, 1990, 2001), Modularity (Baldwin et al., 2009; Clark and Baldwin, 2005) and Design Structure Matrices (DSMs) (Browning, 2001), study architecture of complex systems, which is of particular relevance when dealing with ETD. These theories have allowed researchers to carry out stochastic studies with the aim of predicting how change propagates in complex existing architectures (Clarkson et al., 2004; Sarica and Luo, 2019; Dong et al., 2016; James et al., 2011; D'Amelio et al., 2011). Nevertheless, most of this work is based upon the assumption that we, as designers, are able to dissect, analyse and build models for existing systems. But, unlike the well-documented systems brought forth in the aforementioned papers, some intricate systems are unknown or unknowable. In DevOps frameworks, such systems can experience a progressive accumulation of ETD over time, but since the understanding of their architecture is incomplete, architectural couplings cannot be diagnosed and located. It is in this context that Axiomatic Design comes in handy: we will present a method that shows it can be used as an efficient learning tool, fit to handle an implicit and opaque design issue such as ETD. The core concept is to use the principles of axiomatic design but choose not to dissect a system and treat it as a black box, as opposed to decomposing it and listing explicit interdepecies.

2.4 Research questions

For black box systems in DevOps frameworks, the research questions that we want to inform are:

- RQ 1: To what extent can Axiomatic Design Theory help us diagnose ETD and estimate its impact on the architecture of a modified system ?
- RQ 2: To what degree can Axiomatic Design Theory help us locate and quantify ETD in the architecture of a modified system ?

3 MATERIALS & METHODS

3.1 Methodology choice

In order to answer the RQs, a theoretical model for diagnosing and locating ETD in systems using a black box approach to Axiomatic Design Theory was required; along with an adapted experimental protocol in which to test said model. Our research partnership with Ubisoft's IT branch (that has adopted a DevOps framework) provided a suitable environment to address the RQs.

3.2 Theoretical framework: modeling & analyzing ETD

3.2.1 Axiomatic Design Theory & ETD

As mentioned in subsection 2.3, Axiomatic Design Theory (Suh and Suh, 1990, 2001) provides a formal design framework for our black box analysis of ETD in a given system. This engineering design theory was developed in MIT's department of mechanical engineering by professor N.P Suh in the 1990's. According to the principles of Axiomatic Design Theory, any technical system responds to a series of customer expectations that can be translated into **Functional Requirements (FRs)**, FRs are then fulfilled by a series of **Design Parameters (DPs)**. The correlation between FRs and DPs is systematically studied through Design Matrices, where FRs are represented as rows and DPs as columns. Crosses in Design Matrices indicate the presence of couplings between FRs and DPs. As previously stated, ETD represents an implicit complexification of a system's architecture and an unintentionally coupled design. Therefore, in Axiomatic Design, ETD can be understood as non-diagonal cells in design matrices. Consequently, design matrices provide a practical graphical representation for ETD.

3.2.2 Axiomatic Design Theory & DevOps frameworks

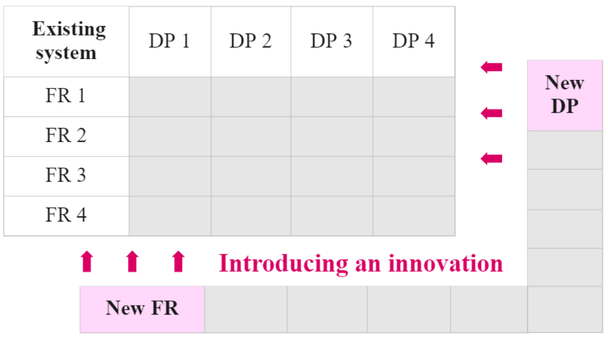


Figure 1. Introducing an innovation into an existing system, can be modeled by the inclusion of a new FR / DP (Row / Column) pair to the system's Design Matrix.

Axiomatic Design Theory provides a suitable model for understanding how new features are implemented in a DevOps framework. In this environment, introducing a new feature into a complex system can be modeled by adding a new row (FR) and its corresponding column (DP) to an existing design matrix, as **Figure 1** shows. However, new features might interact with existing ones through underlying

https://doi.org/10.1017/pds.2023.127 Published online by Cambridge University Press

couplings. If these couplings do in fact exist, introducing a new FR / DP pair into a system can propagate change and disrupt pre-existing FR / DP pairs, which is a well studied phenomenon (Clarkson et al., 2004; James et al., 2011). **Figure 2** illustrates this in an Axiomatic Design framework.

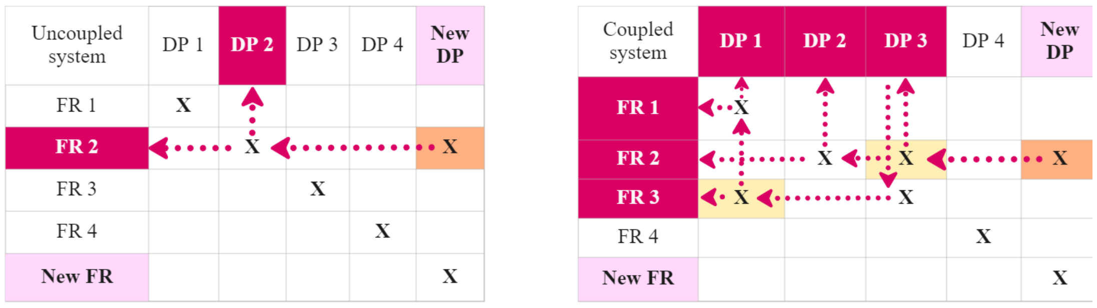


Figure 2. Coupling propagation in decoupled (Left) and coupled (Right) design matrices.

Two possible innovation scenarios are shown: On **Figure 2** (Left) an FR / DP pair (pink cells) with 1 coupling (orange cell) is introduced into a decoupled system. The red arrows represent how this new coupling (orange cell) propagates through the matrix, showing that FR 2, and consequently DP 2, are affected by it. The diagonality of the matrix prevents further propagation. In contrast, **Figure 2** (Right) shows the same FR / DP pair (pink cells) with 1 coupling (orange cell) being introduced into a coupled system. The red arrows show how this new coupling (orange cell) propagates through the matrix, by interacting with underlying couplings (yellow cells) that amplify its effects, resulting in a great disturbance of the system. At first the orange cell indicates that the new DP interacts with FR2. Then, the matrix indicates that FR 2 interacts with DP 2 and DP3. DP 3 interacts with FR 3, which in turn interacts with DP1. At last, DP 1 interacts with FR 1. In conclusion, by following the red arrows, 3 of the 4 existing FR / DP pairs (red cells) are called upon. This example showcases how design matrices can be used as a means of understanding the propagation of couplings in systems. It also clarifies that the impact of introducing a new FR / DP pair depends on the architecture of the pre-existing system: it's in fact the existence of underlying couplings that enables the propagation of new disruptive couplings.

3.2.3 A black box approach to Axiomatic Design Theory & ETD

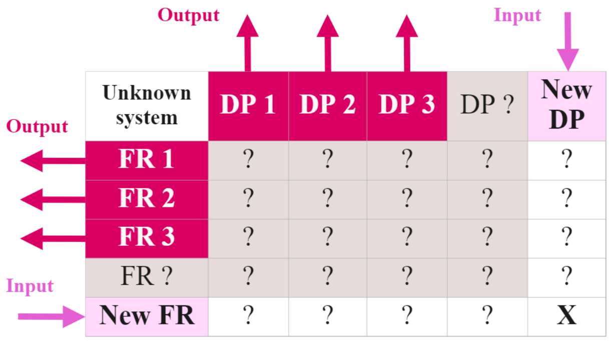


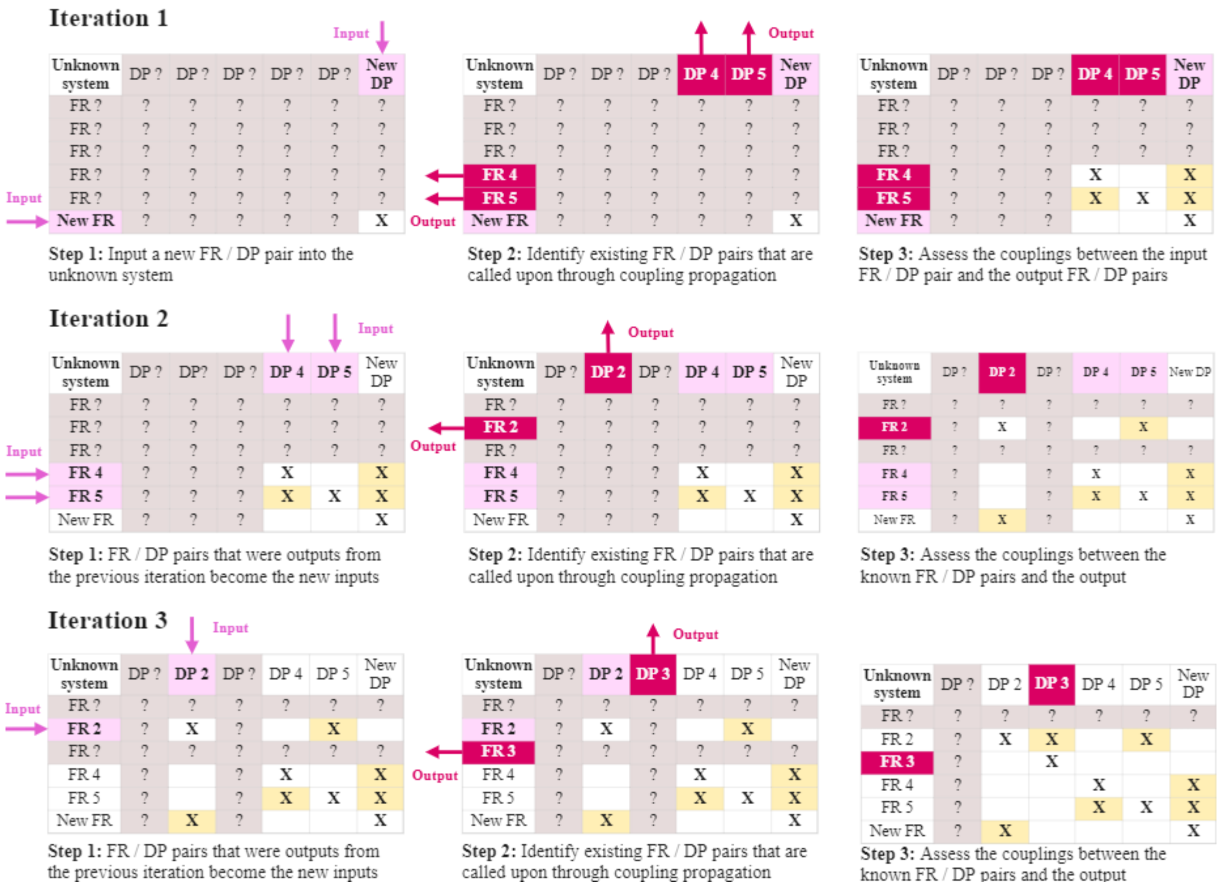
*Figure 3. In Axiomatic Design, approaching an unknown system as a black box corresponds to having a series of matrix cells filled with question marks (gray cells), into which we input a new FR / DP pair (purple arrows). The new FR / DP pair has unknown couplings (white cells) since we do not know how they will interact with unclear existing FRs / DPs. Through a coupling propagation mechanism such as the one depicted in **Figure 2**, a series of FR / DP pairs that are impacted by the change, are called upon (red cells). This partial set of existing FR / DP pairs that were previously unclear, are the outputs of our black box system.*

IT teams rarely possess a complete picture of the system they are designing over. Innovating in such an environment consists in adding new FR / DP pairs to a partially unknown Design Matrix, where some FR / DP pairs are well documented but others are opaque. This incurs a significant risk, since anticipating how new FR / DP pairs might interact with unknown existing pairs is very challenging.

To work within these limitations, a complex system and its Design Matrix can be seen as a black box, where all contents are assumed to be unknown. Then, systems are viewed only in terms of inputs and outputs. As **Figure 3** shows, our black box system is represented by an opaque Design Matrix, filled with question marks. The input is a newly introduced FR / DP pair and the outputs are the pre-existing FR / DP pairs that have interacted with it through the propagation of couplings described in **Figure 2**.

3.2.4 Iterative black box Axiomatic Design as a tool to locate ETD

Real IT systems, with a history of functional expansion and complexification can have crowded design matrices, which are the result of an unclear accumulation of FR / DP pairs. Shedding light on them head-on can be an overwhelming task for teams, as architecture can become exponentially intricate with each DevOps cycle. Therefore a probing strategy to elucidate key elements of these unknown design matrices, based on empirical findings from real Ubisoft IT systems, was conceived. It is named **iterative black box Axiomatic Design**, and is illustrated in **Figure 4**.



Unknown system	DP ?	DP ?	DP ?	DP ?	DP ?	New DP
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
New FR	?	?	?	?	?	X

Step 1: Input a new FR / DP pair into the unknown system

Unknown system	DP ?	DP ?	DP ?	DP 4	DP 5	New DP
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR 4	?	?	?	?	?	?
FR 5	?	?	?	?	?	?
New FR	?	?	?	?	?	X

Step 2: Identify existing FR / DP pairs that are called upon through coupling propagation

Unknown system	DP ?	DP ?	DP ?	DP 4	DP 5	New DP
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR 4	?	?	?	X		X
FR 5	?	?	?	X	X	X
New FR	?	?	?			X

Step 3: Assess the couplings between the input FR / DP pair and the output FR / DP pairs

Unknown system	DP ?	DP?	DP ?	DP 4	DP 5	New DP
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR 4	?	?	?	X		X
FR 5	?	?	?	X	X	X
New FR	?	?	?			X

Step 1: FR / DP pairs that were outputs from the previous iteration become the new inputs

Unknown system	DP ?	DP 2	DP ?	DP 4	DP 5	New DP
FR ?	?	?	?	?	?	?
FR 2	?	?	?	?	?	?
FR ?	?	?	?	?	?	?
FR 4	?	?	?	X		X
FR 5	?	?	?	X	X	X
New FR	?	?	?			X

Step 2: Identify existing FR / DP pairs that are called upon through coupling propagation

Unknown system	DP ?	DP 2	DP ?	DP 4	DP 5	New DP
FR ?	?	?	?	?	?	?
FR 2	?	X	?		X	
FR ?	?	?	?	?	?	?
FR 4	?		?	X		X
FR 5	?		?	X	X	X
New FR	?	X	?			X

Step 3: Assess the couplings between the known FR / DP pairs and the output

Unknown system	DP ?	DP 2	DP ?	DP 4	DP 5	New DP
FR ?	?	?	?	?	?	?
FR 2	?	X	?		X	
FR ?	?	?	?	?	?	?
FR 4	?		?	X		X
FR 5	?		?	X	X	X
New FR	?	X	?			X

Step 1: FR / DP pairs that were outputs from the previous iteration become the new inputs

Unknown system	DP ?	DP 2	DP 3	DP 4	DP 5	New DP
FR ?	?	?	?	?	?	?
FR 2	?	X	?		X	
FR 3	?	?	?	?	?	?
FR 4	?		?	X		X
FR 5	?		?	X	X	X
New FR	?	X	?			X

Step 2: Identify existing FR / DP pairs that are called upon through coupling propagation

Unknown system	DP ?	DP 2	DP 3	DP 4	DP 5	New DP
FR ?	?	?	?	?	?	?
FR 2	?	X	X		X	
FR 3	?		X			
FR 4	?			X		X
FR 5	?			X	X	X
New FR	?	X				X

Step 3: Assess the couplings between the known FR / DP pairs and the output

Figure 4. Illustration of the iterative black box Axiomatic Design process.

The starting point is an unknown system, on which we introduce a new FR / DP pair, seen as the input of the black box system, highlighted in pink (**Figure 4 - Iteration 1 - Step 1**). The introduction of the new FR / DP pair kick-starts the coupling propagation mechanism explained in **Figure 2**, and calls upon pre-existing FR / DP pairs that have been impacted by the change, represented as outputs in red (**Figure 4 - Iteration 1 - Step 2**). It is at this point that the apparent opacity of the system can be overcome, since this method puts FRs at the center. FRs are user-related by nature and therefore easier to express by users, whenever a new FR conflicts with an existing one users will notify it. This process builds a list of pre-existing FRs which can then be attributed to DPs with the help of system architects and technical experts. Once these pre-existing FR / DP pairs become known, it is possible to determine how they interact with the rest of known FR / DP pairs. This is represented by question mark cells in the matrix becoming clear. If there is no coupling they are blank white cells, if there is an unwanted coupling they are highlighted in yellow (**Figure 4 - Iteration 1 - Step 3**). This process can then be iterated to

https://doi.org/10.1017/pds.2023.127 Published online by Cambridge University Press

uncover more FR / DP pairs. For the next iteration, shown in **Figure 4 -Iteration 2**, the outputs from the previous iteration are taken as inputs, and the same 3 step procedure is followed. The final step of the third iteration (**Figure 4, Iteration 3**) shows the result of the total accumulated knowledge from every iteration of this process: a clearer, but partial, Design Matrix, with a series of well defined FR / DP pairs and explicit couplings. As stated, knowledge of the system remains partial, since it is only possible to gain information about FR / DP pairs if they interact with each other through coupling propagation. Therefore, some FR / DP pairs are still opaque, as evidenced by the remaining question marks in **Figure 4, Iteration 3, Step 3**.

3.2.5 A complementary tool: the ETD diagnosis chart

The model described in the previous subsection is accurate but time-consuming for teams to implement on a wide range of systems. Moreover, technical documentation available at Ubisoft does not take the form of explicit FRs and DPs. Therefore, a complementary approach adapted to our corporate environment needed to be developed, which is showcased in **Figure 5**.

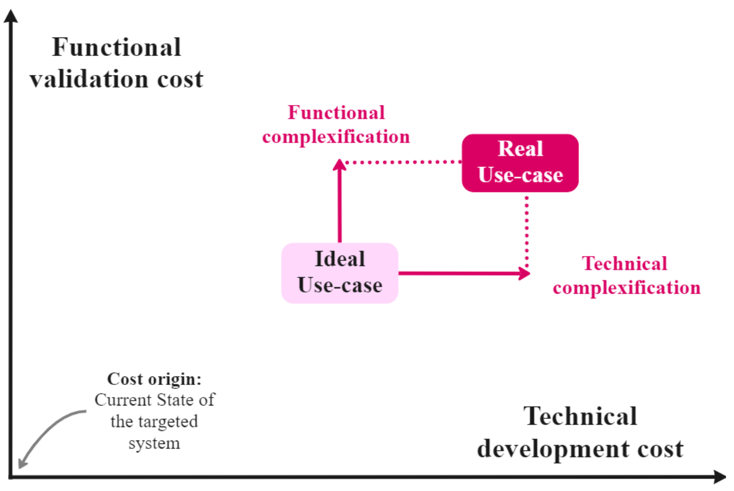

Functional validation cost
Functional complexification
Real Use-case
Ideal Use-case
Technical complexification
Cost origin: Current State of the targeted system
Technical development cost

Figure 5. ETD diagnosis chart. The placement of the pink rectangle indicates the functional validation cost and the technical development cost of an ideal ("in vitro") version of a use-case, with no ETD. Red vectors indicate complexification along both axes after the effects of ETD have been taken into account and depict the real ("in vivo") cost of a use-case.

Here, FR / DP were reworded into functional validation / technical development and were assessed through a "cost" lens. Added cost is merely a symptom of ETD but it has the advantage of being well documented and it brings a sense of urgency, which eased communication with corporate management structures. To break down both axes: **Technical development** cost, reflects the cost of a successful implementation of the new DPs related to the use-case at hand (→ Analogy: number of new lines of code that need to be written). **Functional validation** cost corresponds to the validation effort that needs to be carried out in order to adequately test the new FRs related to the use-case at hand (→ Analogy: number of tests that the new feature needs to pass). **Figure 5** also provides an assessment of a project's potential complexification, which is the "added" cost caused by ETD, broken down using the same two axes. **Technical complexification** can be understood as the number of pre-existing DPs that are impacted by the implementation of the use-case at hand (→ Analogy: number of legacy lines of code that require refactoring). **Functional complexification** corresponds to the number pre-existing FRs that are impacted by the implementation of the use-case at hand (→ Analogy: number of new tests pre-existing features need to pass). In a nutshell, **Figure 5** conveys the fact that a use-case has an "in vitro" cost, and an "in vivo" cost, that has been amplified by the existence of couplings in architecture. The additional cost often takes the form of refactoring efforts, and is a characteristic symptom of ETD in systems.

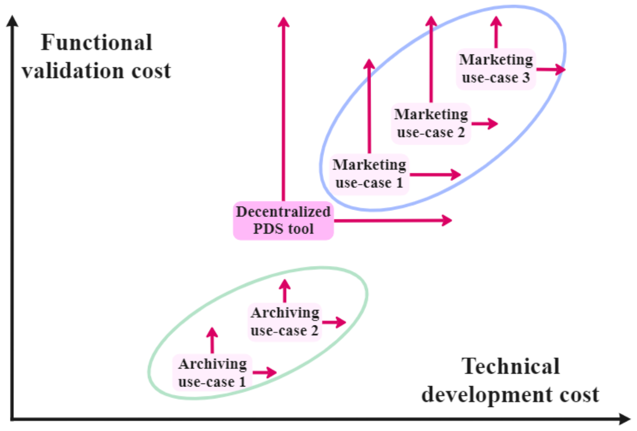
3.3 Empirical material

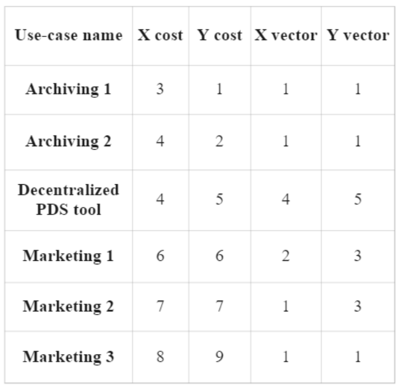
The empirical material consists of 6 ongoing innovative IT projects from Ubisoft's IT portfolio, that we will refer to as use-cases. All of the studied use-cases revolved around a common theme: decentralized data storage. One of these 6 decentralized storage use-cases was selected, it consisted in adding a new FR / DP pair to an existing IT system, that we will refer to as the Production Data Sharing tool (PDS tool) for confidentiality purposes. It is a system that allows the mutualization of game assets between different developer machines. The PDS tool adequately met the needs of developers, but following the Covid-19 crisis a new Work from Home FR arose. To tackle this new FR, the introduction of a new DP into the PDS tool, in the form of a peer to peer network based on IPFS (Benet, 2014), was explored. From a quantitative standpoint, the empirical material consists of an archive review (**12** documents reviewed, **83** pages of written documents, **195** minutes of audiovisual documents) and meetings & workshops (**25** experts consulted, **179** pages of meeting notes and **106** hours spent on dedicated interviews). The job descriptions of the consulted experts comprise, IT directors, IT Project managers, Storage Architects, DevOps engineers, Blockchain Technical Leads, Business Analysts and Design engineering Professors.

4 RESULTS

4.1 Diagnosing ETD

Using the use-case ETD diagnosis chart, six decentralized storage use-cases from Ubisoft's IT portfolio were assessed. **Figure 6** shows their placement on the chart along with their complexification vectors,





Use-case name	X cost	Y cost	X vector	Y vector
Archiving 1	3	1	1	1
Archiving 2	4	2	1	1
Decentralized PDS tool	4	5	4	5
Marketing 1	6	6	2	3
Marketing 2	7	7	1	3
Marketing 3	8	9	1	1

Figure 6. Left: ETD diagnosis chart for decentralized storage use-cases. Right: use-case coordinates on a qualitative scale of 1 to 10

These results allow us to draw conclusions regarding DevOps's ability to handle exploratory use-cases with ETD symptoms. In traditional waterfall approaches, business teams set a functional validation target and then development teams carry out the technical work required, meaning the two axes of **Figure 5**, technical development and functional validation are purposely decorrelated. In contrast, DevOps fosters incremental projects, amplifying feedback loops between functional validation and technical development. This is of particular interest for use-cases with a high cost need, which need to be broken down into short, incremental steps. If said intermediate steps are not clearly defined, good practices, such as adequate release and test schedules, are lost in spite of DevOps' focus on systematic feedback integration. The need to establish intermediate steps is explicit for highly exploratory use-cases, with a high ideal cost, such as the ones circled in blue in **Figure 6**. However, certain use-cases possess a manageable ideal cost but a greatly augmented real cost, which can trump teams into believing that a DevOps approach might suffice, which is the case of the decentralized PDS tool use-case, sitting in the center of **Figure 6**. When the effects of ETD are taken into account and cost augments, it becomes apparent that this use-case needs to be broken down into shorter steps. DevOps frameworks without a dedicated ETD management approach fail to provide a healthy environment to manage such use-cases. Consequently, a complementary method to understand ETD, accurately assess its effects, and break use-cases down into incremental steps is required. Hence the need for Iterative black box Axiomatic Design.

4.2 Locating ETD

Iterative black box Axiomatic Design was tested on the Decentralized PDS tool. In this use-case, new features that allow for adequate Work from Home conditions were introduced into the system. **Figure 7** shows the PDS tool Design Matrix, indicating that the new FR / DP pair brings 4 new couplings (orange cells) to a design matrix which already possessed 6 couplings (yellow cells). Through this analysis ETD in the PDS tool has been clearly located. Among the 10 couplings shown in **Figure 7**, one focuses on ensuring corporate cybersecurity standards when using IPFS. For this coupling, deemed critical, a strategy built around the configuration, deployment and maintenance of user permissions and node identities was put in motion by linking actors and facilitating the collaboration of Ubisoft's IT teams.

PDS tool	DP 1	DP 2	DP 3	DP 4	DP 5
FR 1	**X**			X	
FR 2	X	**X**	X	X	X
FR 3			**X**	X	X
FR 4		X		**X**	X
FR 5			X		**X**

Note:

Design matrices and all related information have been altered and are given for illustration purposes only.

Consequently FRs and DPs will not be explicitely stated.

Figure 7. PDS tool design matrix.

5 DISCUSSION

5.1 Paper contributions

This paper provides a formal design framework for the analysis of ETD in systems, it brings forth a graphical representation in the form of design matrices, and a means to understand how ETD propagates and amplifies the cost of new features. To complement this theoretical study, an ETD diagnosis method was developed. Use-cases are assessed by being placed on the ETD diagnosis chart, which shows their inherent cost and also takes into account the complexification caused by ETD, which amplifies said cost. In order to locate ETD in specific systems, the Iterative black box Axiomatic Design method was proposed, progressively uncovering FRs and DPs of a system and identifying their couplings. In doing so, it allows teams to pinpoint critical couplings. This knowledge can then contribute to establishing a prioritization guide for subsequent evolutions, by naturally breaking down ambitious exploratory use-cases into incremental steps, therefore generating innovation trajectories that suit DevOps frameworks.

5.2 Future work & paper limits

The use-cases treated in **Figure 6** can be split into two categories, which provides us with a glimpse into future interesting perspectives. On one hand, use-cases with a low cost and low complexification level, with clear deliverables (green circle in **Figure 6**). On the other hand, costly exploratory use-cases which possess deliverables that are more difficult to define, and present a greater complexification potential (blue circle in **Figure 6**). This separation allows us to define a DevOps compatibility zone. It is the zone inside which use-cases can be considered as individual incremental steps of an innovation trajectory, fit to be handled by a DevOps framework. In this zone complexification is contained, and ETD does not cause a propagation of couplings that can make a use-case cross the compatibility threshold. The existence of such a zone is hinted at, but its characteristics or its borders are not precisely defined. Digging deeper into these concepts will constitute a natural progression to this paper. Furthermore, the result of an iterative black box Axiomatic Design analysis is a partial Design Matrix. The information is partial on two different fronts: the Design Matrix locates couplings but does not provide information regarding their nature or a quantification of their complexity. The Design Matrix does not showcase an exhaustive set of FR / DP pairs, so it is possible that FR / DP pairs exist in a silo, untouched by the coupling propagation mechanism, and invisible to iterative black box Axiomatic Design. Building on the contents of this paper to develop a method that enables an accurate quantification of a coupling's complexity, or that maps architectural silos inside a system, is the logical continuation of our work.

https://doi.org/10.1017/pds.2023.127 Published online by Cambridge University Press

REFERENCES

Baldwin, C.Y., Woodard, C.J. et al. (2009), "The architecture of platforms: A unified view", *Platforms, markets and innovation*, Vol. 32, pp. 19–44, http://doi.org/10.4337/9781849803311.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R. et al. (2001), "Manifesto for agile software development", Available at: http://agilemanifesto.org/.

Benet, J. (2014), "Ipfs-content addressed, versioned, p2p file system", *arXiv preprint arXiv:1407.3561*, http://doi.org/10.48550/arXiv.1407.3561.

Browning, T.R. (2001), "Applying the design structure matrix to system decomposition and integration problems: a review and new directions", *IEEE Transactions on Engineering management*, Vol. 48 No. 3, pp. 292–306, http://doi.org/10.1109/17.946528.

Clark, K.B. and Baldwin, C.Y. (2005), "Designs and design architecture: The missing link between'knowledge'and the'economy"', http://doi.org/10.2139/ssrn.664043.

Clarkson, P.J., Simons, C. and Eckert, C. (2004), "Predicting change propagation in complex design", *J. Mech. Des.*, Vol. 126 No. 5, pp. 788–797, http://doi.org/10.1115/1.1765117.

Colomo-Palacios, R. et al. (2020), "Continuous practices and technical debt: a systematic literature review", in: *2020 20th International Conference on Computational Science and Its Applications (ICCSA)*, IEEE, pp. 40–44, http://doi.org/10.1109/iccsa50381.2020.00018.

Cunningham, W. (1992), "The wycash portfolio management system", *ACM SIGPLAN OOPS Messenger*, Vol. 4 No. 2, pp. 29–30, http://doi.org/10.1145/157710.157715.

D'Amelio, V., Chmarra, M.K. and Tomiyama, T. (2011), "Early design interference detection based on qualitative physics", *Research in Engineering Design*, Vol. 22, pp. 223–243, http://doi.org/10.1007/s00163-011-0108-7.

Dong, A., Sarkar, S., Moullec, M.L. and Jankovic, M. (2016), "Eigenvector rotation as an estimation of architectural change", in: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 50190, American Society of Mechanical Engineers, p. V007T06A014, http://doi.org/10.1115/detc2016-59114.

Fowler, M. (2009), "Technical debt quadrant", *Martin Fowler*, pp. 14–0.

James, D., Sinha, K. and de Weck, O. (2011), "Technology insertion in turbofan engine and assessment of architectural complexity", in: *DSM 2011: Proceedings of the 13th International DSM Conference*.

Li, Z., Avgeriou, P. and Liang, P. (2015), "A systematic mapping study on technical debt and its management", *Journal of Systems and Software*, Vol. 101, pp. 193–220, http://doi.org/10.1016/j.jss.2014.12.027.

Liodden, D. (2020), "3 main types of technical debt and how to manage them", in: *annual CTO summit*, FirstMark.

Martini, A., Bosch, J. and Chaudron, M. (2014), "Architecture technical debt: Understanding causes and a qualitative model", in: *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, IEEE, pp. 85–92, http://doi.org/10.1109/seaa.2014.65.

McConnell, S. (2007), "Technical debt", *Software Best Practices, Nov*.

Ohno, T. (2019), *Toyota production system: beyond large-scale production*, Productivity press, http://doi.org/10.4324/9780429273018.

Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019), "A systematic mapping study of infrastructure as code research", *Information and Software Technology*, Vol. 108, pp. 65–77, http://doi.org/10.1016/j.infsof.2018.12.004.

Sarica, S. and Luo, J. (2019), "An infinite regress model of design change propagation in complex systems", *IEEE systems journal*, Vol. 13 No. 4, pp. 3610–3618, http://doi.org/10.1109/jsyst.2019.2899988.

Shahin, M., Babar, M.A. and Zhu, L. (2017), "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices", *IEEE Access*, Vol. 5, pp. 3909–3943, http://doi.org/10.1109/ACCESS.2017.2685629.

Suh, N.P. and Suh, N.P. (2001), *Axiomatic design: advances and applications*, Vol. 4, Oxford university press New York.

Suh, N.P. and Suh, P.N. (1990), *The principles of design*, 6, Oxford University Press on Demand.

Zhu, L., Bass, L. and Champlin-Scharff, G. (2016), "Devops and its practices", *IEEE Software*, Vol. 33 No. 3, pp. 32–34, http://doi.org/10.1109/ms.2016.81.