Redefining Legacy: A Technical Debt Perspective

Ben D. Monaghan[1][0000−0001−6755−0304] and Julian M. Bass[1][0000−0002−0570−7086]

University Of Salford, 43 Crescent, Salford M5 4WT, UK
B.D.Monaghan1@edu.salford.ac.uk
J.Bass@salford.ac.uk

Abstract. Organisations that manage legacy systems at scale, such as those found within large government agencies and commercial enterprises, face a set of unique challenges. They manage complex software landscapes that have evolved over decades. Current conceptual definitions of legacy systems give practitioners limited insights that can inform their daily work. In this research, we compare conceptual definitions of large-scale legacy and technical debt. We hypothesise that large-scale legacy reflects an accumulation of technical debt that has never been through a remediation phase. To pursue this hypothesis, we identified the following question: *How do practitioners describe their experience of managing large-scale legacy landscapes?* We conducted 16 semi-structured open-ended, recorded and transcribed interviews with industry practitioners from 4 government organisations and 9 large enterprises involved with the maintenance and migration of large-scale legacy systems. A snowball sampling technique was used to identify participants. We adopted an approach informed by grounded theory. There was consensus among the practitioners in our study that the landscape is fragmented and inflexible, consisting of many dispersed and fragile applications. Practitioners report challenges with shifting paradigms from batch processing to near real-time customer-focused information systems. Our findings show there is overlap between challenges experienced by participants and symptoms typified by technical debt. We identify a novel type of technical debt, "Ecosystem Debt" which arises from the scale, and age, of many large-scale legacy applications. By positioning Legacy within the context of Technical Debt, practitioners have a more concrete understanding of the state of the systems they maintain.

Keywords: Legacy Software · Technical Debt · Software Evolution · Industry Perspectives · Ecosystem Debt · Software Ecosystems

1 Introduction

Legacy Software [4, 5] is everywhere. From local companies to tech giants, it is an issue every industry faces. Despite the issues (and costs) being well known for decades, it's a problem which has persisted. One area where legacy is particularly

persistent is within governments and large enterprises. The definition of Legacy Software varies. It has been defined as software that is outdated and old [4], or, software that is mission critical but brittle, expensive to maintain and resistant to changes [5].

Legacy Systems are typically associated with mainframe-based languages such as COBOL and Fortran [12]. This, however, is not always the case. Modern software developed using new techniques can also satisfy the criteria of legacy [22]. This is especially true in large-scale web development where a product has a long development time before it goes to market and the latest web framework is now outdated. One common thread throughout however is that they are mission critical and therefore maintenance costs must be tolerated.

Challenges to managing and maintaining legacy software are well understood [5, 12]. It is difficult to maintain and difficult to port to new technologies. Often legacy software is heavily integrated with the physical hardware it operates in (i.e. mainframes). However, legacy software persists. It has been estimated that there are still billions of lines of legacy code in use [16]. This raises the question of why are companies unwilling, or unable, to modernise their legacy applications? We posit that current conceptual definitions of legacy fail to accurately convey the day to day challenges faced when managing legacy systems. To this end, we propose an alternative way of defining legacy by viewing it from the perspective of technical debt. However, in order to understand how legacy relates to technical debt we first need to capture the experiences of industry practitioners involved in the maintenance and migration of legacy systems. This leads to the question: *How do practitioners describe their experience of managing large-scale legacy landscapes?*

In this paper we adopt a grounded theory approach, analysing 16 open-ended, transcribed interviews from industry practitioners who have been involved in the maintenance or migration of large scale legacy systems. We identify common characteristics of large scale legacy systems. We present a taxonomy mapping these to types of technical debt. We propose a new type of technical debt, which we refer to as "Ecosystem Debt". We also extend existing definitions of technical debt to account for the impact of age on code related debt. Specifically in relation to the cognitive gap which arises from shifts in development paradigms. The rest of this paper is organised as follows: the next section provides the background. Sect. 3 describes the research methods used. In Sect. 4 we present our findings. In Sect. 5 we discuss those findings any threats to validity. Finally we conclude in Sect. 6

2 Background

2.1 Legacy Systems

The definition of Legacy Software varies. They are defined as software that is outdated, or old, software that is mission critical but brittle, expensive to maintain and resistant to changes [4]. One definition proposed by [8] is that they

simply belong to a previous generation of technology. This, however, isn't always the case. Modern software developed using new techniques can also satisfy the criteria of legacy [17]. This is especially true in modern web development where if a product has a particularly long development time before it goes to market, what's modern has probably evolved and the latest web framework is now old. One common thread throughout however is that they are mission critical and therefore worth the costs associated with maintaining them.

Research into legacy modernisation typically focuses on technical challenges, with little focus on industry perceptions. Research often runs on the assumption that legacy software is obsolete. However there is evidence that this may not always be the case [16]. Practitioners from a spectrum of positions and fields were interviewed about their perception of what their experiences with legacy software. Their results echo existing views regarding the challenges associated with legacy software, however they also reveal that to a number of respondents held a favourable view of legacy software. There is a perception that legacy software is proven technology, reliable and perhaps counter intuitively, performant.

2.2 Technical Debt

The term Technical Debt (TD) was first introduced by Cunningham [7]. TD is used to describe developing poor-quality systems for short term gain (often for expedience), with the view that at some point in the future the work will need to be revisited. TD, much like financial debt, can bring benefits in the short term (on the provision that is paid back promptly).

Although the term debt might be viewed as a bad thing, this is not necessarily the case. Going into TD can be part of a larger strategic decision to bring a product to market quicker, delaying quality and robustness until further into the future [24].

While accruing TD may be a strategic decision, much like conventional debt, it needs to be kept under control and managed. Failure to do so results in brittle, hard to maintain software that becomes costly and difficult to comprehend [26]. In certain cases the build-up of TD may not be a conscious decision [6]. Martin Fowler suggests breaking down reasons for TD into reckless, prudent, inadvertent and deliberate [10]. There are also types of TD, such as code related or architectural debt, Alves et al present an ontology where they identify 12 different types [1]. Similar types are identified by both Kutchens et al [19] and Rios et al [23].

2.3 Relationship Between Technical Debt and Legacy

Legacy software and Technical debt share similarities in that they are both perceived to mean software that is in a poor state or of low quality. Holvitie et al [15] explore the closeness of technical debt and legacy software. They present conceptual definitions of both, highlighting the similarities and that, depending on the context and interpretation both terms can be used to describe the same symptoms.

Technical debt also offers a potential mechanism for improving the quality and longevity of existing legacy software. Gupta et al [14] present a case study on managing a legacy application by tackling technical debt issues. They show a decrease in a number of quality defects, such as memory issues, system crashes and performance related issues, suggesting that approaching legacy from a technical debt perspective can alleviate issues commonly associated with ageing software.

2.4 Software Ecosystems

Definitions of Software Ecosystems (SECO) vary [11]. Efforts have been made to create a more concrete definition by [20], they provide the definition of a SECO '*as the interaction of a set of actors on top of a common technological platform that results in a number of software solutions or services*'.

Research into SECOs is a growing area, however we note that there is little research into the impact of technical debt and SECOs, much of existing research focuses on the context and ecosystem health [11]. McGregor et al [21] present software ecosystems within the context of technical debt, they highlight that the effects of technical debt in one aspect of a software ecosystem can have impacts on other components within the same ecosystem.

3 Method

In this paper we analyse industry practitioner experiences when managing large scale legacy software. The research question we answer is: *How do practitioners describe their experience of managing large-scale legacy landscapes?*.

A qualitative method approach was adopted in this study to capture and analyse industry practitioner experiences within the context of large scale legacy software. We adopted a Grounded Theory (GT) approach, analysing data collected from semi-structured open ended interviews. GT was chosen to avoid preconceived assumptions about how legacy is maintained and perceived by practitioners.

3.1 Research Sites

A mix of practitioners from a variety of backgrounds and industries were identified using a snowball sampling technique. Initial participants were identified through the authors professional contact network, subsequent participants were then identified on recommendation by initial participants. The criteria for selection was to be, or have been, involved in managing or maintaining large scale legacy systems. Interviews were conducted both in person through meetings and where that was not possible via Skype. The participants in this study are listed in table 1. To highlight the scale of the organisations that participants were from, we briefly describe P1, Major City Council and P9, Large Insurance Company.

P1, Major City Council. P1 Is an Enterprise Architect for a Major UK City Council (population >400k), which employs >15,000 staff. P1 is involved in

ensuring ICT systems across the City Council are aligned to the overall business strategy.

P9, Large Insurance Company. P9 Is an IT Development manager for a large UK based Insurance company. They have a revenue of >£5 Billion and employ >2000 staff. P9 is involved in managing the legacy estate that the organisation maintains, they operate out of a business unit specifically designed to manage large scale legacy within the organisation.

Table 1. List of Research Participants

Identifier	Job Title	Industry	Experience
P1	Enterprise Architect	Major City Council	18 years
P2	CTO	Start-up	15+ years
P3	IT Director	High Street Retailer	30 years
P4	CIO	Government Agency	15+ Years
P5	Head of Project Delivery	Government Department	15+ Years
P6	CIO	Banking	40+ Years
P7	Senior Delivery Leader	Government Department	40+ Years
P8	CIO	Large Enterprise	40+ Years
P9	IT Development Manager	Large Insurance Company	40+ Years
P10	CIO	High Street Retailer	30+ Years
P11	Lead DBA	Regional Energy Company	17 Years
P12	CIO	Consultancy Company	25+ Years
P13	Head of Software Engineering	Government Agency	16 Years
P14	CIO	High Street Retail & Banking	40 Years
P15	Principal Architect	Government Agency	25+ Years
P16	Digital Directory	Government Department	30 Years

3.2 Data Collection

A total of 16 semi-structured interviews [3] were conducted for data collection. An open-ended approach was adopted to allow the interviewee a chance to cover any other issues of interest beyond the semi-structured interviews. Interviews were conducted in person where possible, otherwise remotely via Skype and recorded. The interview recordings were then transcribed by hand. Questions were revisited after each interview and refined. Each interview continued for between 45-60 minutes.

3.3 Data Analysis

3.4 Interviews

For this paper we adopted a classical grounded theory (Glasserian) approach [13]. Grounded Theory aims to develop a theory from data without any pre-conceived

perceptions. To enable this approach Interview transcripts were analysed using an industry standard qualitative analysis tool *nVivo*. We first went through the initial transcripts, using a line by line coding. As we began to understand the data we grouped codes into category, this allowed us to identify recurring topics within the interview transcripts. These categories were then further grouped into concepts. Each concept was analysed using memos. Memos were iteratively refined via constant comparison of the concepts with the raw data.

3.5 Classifying Legacy in Terms of Technical Debt

In order to classify legacy in terms of technical debt, we identified the major types of technical debt from literature. We limited our search to 2015-2020, using the keywords *"Technical Debt Types" OR "Technical Debt Dimensions" 'OR "Technical Debt Categories"*. Digital libraries considered for this search were ACM Digital Library, IEEE Xplore, Science Direct and Springer Link. Selection criteria was defined as papers which define or describe types/dimensions/categories of Technical Debt. We identified three main sources which provided definitions of technical debt types; Alves et al, [1], Krutchen et al [19] and Rios et al [23]. For each concept generated as a part of interview data analysis, we compared participants descriptions of symptoms/difficulties that they have to manage with those described in literature to see how much, or if any, overlap there was.

4 Findings

The following section presents our findings and is structured as follows; Subsection 4.1 presents how respondents viewed and defined legacy systems. Subsection 4.2 presents the results of our analysis of practitioner experiences as compared with Technical debt types.The remaining Subsections (4.3 through to 4.8) present the the interview responses that formed the concepts highlighted in table 2. Figure 1 presents an example of how this process was applied to produce the concept *Growing Skills Gaps Impacts on ability to maintain and evolve*, the findings that support this concept are presented in subsection 4.7.

4.1 Practitioner Understanding of Legacy

Participants describe legacy as old code on old hardware, *"They tend to be older systems developed with older technology. Typically older programming languages, or even sitting on older hardware"* - P1, Enterprise Architect Major City Council. And potentially no longer supported by vendors, *"Coupled with that, potentially, the vendors no longer support the products as well"* - P1, Enterprise Architect Major City Council.

Interviews reveal that legacy systems are at the end of their useful life, *"The legacy system has been in place 10 or 15 years. It's at the end of its useful life now"* - P5, Head of Project delivery, Government Department. This is echoed by P14, CIO High Street Retailer & Bank, *"Sometimes, when people talk about*

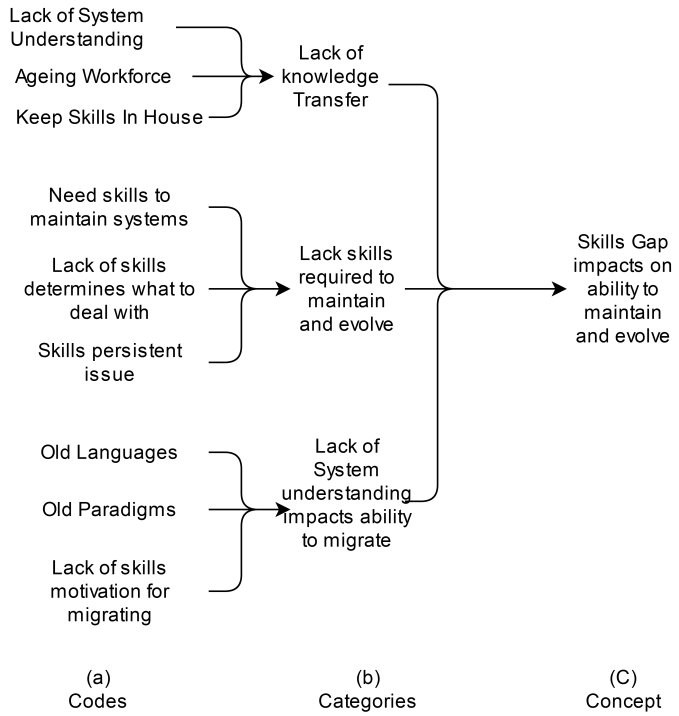

Fig. 1. Illustration on process of iterating through codes, categories and concepts

legacy they are talking about things that are nearing end of life". However P14 also suggests that legacy is simply something which has been there for a long time, *"sometimes when people talk about legacy they just mean things that have been there a long time, but it still does the job, it still works."*. P14 goes onto provide a more concrete definition within their organisation, *"So, in [Practitioners Organisation], when people talk about legacy systems, they talk about the things that they want to replace, they want to get rid of"*.

When discussing the term legacy, P16 suggests *"I sort of try not to use the term heritage and legacy. They have a certain connotation, which is these are old things that we shouldn't have"* P16, Digital Director, Government Department. They go on to then suggest that the driving force should be meeting business needs, *"when you look forward you look at; what is your business looking to achieve?"* - P16, Digital Director Government Department. Respondents describe the importance of these systems to business value, *"We have many systems that deliver over £100b... to the public running on COBOL-based VME"* – Digital Director, Government Department. They need to support large numbers of customers, *"they had about 4,000,000 customers"* CIO, High Street Retail, that represents significant value *"average daily sales was about £2,000,000"* - CIO, High Street Retail.

4.2 Legacy as a Product of Technical Debt

The concepts that were formed from interview analysis were compared with definitions of technical available in literature [1, 19, 23], we present the results of this analysis in Table 2.

Table 2. Taxonomy Classifying Participants Experience with Legacy to Types of Technical Debt

Legacy Concept	*Symptoms*	*TD type*
Legacy Applications support surrounding systems	Difficult to modify due to external dependencies	Ecosystem Debt
Applications are fragmented	Unforeseen consequences when modifying or removing legacy	Ecosystem Debt
Inherited Legacy	Difficult to evolve, maintain and integrate	Ecosystem Debt, Code Debt Architectural Debt Infrastructure Debt Design Debt, Build Debt, Test Debt Documentation Debt
Applications no longer represent the organisation needs	System Architecture no longer supports business needs Code quality has eroded over time impacting maintainability	Code Debt Architectural Debt
Skills Gap impacts ability to maintain or evolve	Challenges understanding system. Developer premiums. Fear of modifying underlying system. Lack of maintainability and difficult to evolve.	Code Debt, Design Debt Architectural Debt Documentation Debt
Complex System Architectures	Difficult to maintain or evolve New functionality is bolted on Architectural Drift	Code Debt, Architectural Debt

4.3 Legacy Applications Support Surrounding Systems

The interviews reveal that large-scale legacy systems are composed of significant numbers of applications, *"in the retail bank you had over 600 applications"* - P12 CIO Consultancy Company. P13 describes a large number of product platforms *"in the Cobol estate there are 29 products platforms"* – P13 Head of SWE

Government Department, a similar situation is also confirmed by P2, *"There are quite a lot of applications"* – P2, CTO of Start-up.

Participants describe the size and scale of the legacy systems they are managing, *"One is, we've got a legacy fraud system. It's the system that we use for all of our fraud casework here. Bearing in mind, we're a big organisation – 80,000 people – so fraud is a big deal to us."* – Head of Project Delivery, Government Department. And need to support large numbers of internal operations *"I look after all our contact centre solutions. We have 30,000 contact centre seats here"* – P16, Director of Digital Platforms, Government Department.

Respondents describe how they need to manage many independent applications. These applications support large numbers of internal and external users. They describe how they process significant numbers of transactions, both in number, and in value.

4.4 Applications are Fragmented

Participants describe highly dispersed and fragmented application landscapes, *"The application landscape is highly fragmented. There are a lot of dispersed systems that aren't necessarily connected together"* – P2, CTO of Start-up.

P2 goes on to describe how individual applications support surrounding systems, *"that system had been there so long it was supported by a number of other surrounding systems as well."* – P2, CTO of Start-up. They describe that this them difficult to replace as they impact the wider functionality of a system, *"Simply replacing the legacy system wouldn't necessarily solve all of our problems. We'd be impacting other parts of the process as well"* – P2, CTO of Start-up. In one example, a single application was interfacing with upwards of 200 other systems.

"For example, one of the legacy systems I've replaced, I think it had interfaces with about 200 other systems. So you've got this real jigsaw of all these different systems that link together" – P5, Head of Project Delivery, Government Department.

Respondents describe a highly fragmented application landscape. These applications can be disconnected and dispersed, or part of a complex jigsaw of dependencies. Migrating or replacing these legacy applications can be a challenge.

4.5 Inherited Legacy

The interviews reveal a number of participants have inherited applications when companies are bought up, *"So at the moment it's a complex collection of legacy systems, some of which even came into the [High Street Retailer] when we acquired the [High Street Retail] a very long time ago"* – P10, CIO, High Street Retail. P9 describes a similar scenario, *"we have really big acquired insurance*

businesses which all come with their own large large legacy estates" - P9, IT Development Manager, Large Enterprise. One respondent, when describing acquiring a smaller company highlighted the impact this can have,

"they had to go through all of these scripts, either, switching them off and seeing what happened or trying to put logging into them to try to see if and when they got touched... just monitoring the whole thing to try to gradually unpick this delicate, fragile, kind of landscape of scripts" – P2, CTO of Start-up.

Respondents describe acquiring smaller companies. As part of this acquisition they inherit the pre-existing systems that support the newly acquired company. This expands their existing landscape even further. In some cases, the applications they inherit may be problematic.

4.6 Applications No Longer Represent the Organisation's Needs

Participants describe how a shift in user expectation drives evolution, *"A lot of people expect to interact with us, as an organisation – and many other organisations – in the way that they do with the likes of the Amazon platforms."* - P1, Enterprise Architect, Major City Council. One participant describes *"the organisation is transforming quite significantly, we've got lots of new policy measures coming in, new ways of working"* - P5, Head of Project Delivery, Government Department. The same participant goes on to say

"Because the organisation is transforming quite significantly, we've got lots of new policy measures coming in, new ways of working. I think we're also moving towards a more digital organisation. It means that some of our legacy systems just don't work in the new world", and that *"In terms of what our users expect, so citizens – but also some of our business processes – the current systems just don't work"* - P5, Head of Project Delivery, Government Department.

Participants from Government departments describe organisation needs are driven by driven by policy change, *"we have policy units that take the government legislation, interpret it into what it is intended to do; that is translated into, "Right. So, we need to change what we actually provide to our customers." So, the systems will change underneath."* – P7, Senior Delivery Leader, Government Department. This is compounded by the frequency in which government policy changes, *"They change all the time, because government policy changes."* - P7, Senior Delivery Leader ,Government Department

One participants reveals that attempts to evolve with frequently policy driven needs results in the degradation of the system, *"... we tend to append technology so if we started off with a nice and clean nucleus of an estate 20 or 30 years ago is essentially we've gone right we need to implement this new policy and we will add that on and add that"* – P13, Head of SWE, Government Department.

Participants describe their legacy systems as no longer being able to support organisation needs. Those from government departments describe the need to

keep up with frequent shift in government policy causing systems to degrade over time.

4.7 Skills Gap Impacts Ability to Maintain or Evolve

Participants describe how Legacy systems can be many years old *"so the majority of our systems are on VME so some of these VME machines have been around maybe something like 40 years"* – Principal Architect, Government Department. These systems reflect a different world, *"These systems reflect what the world was, maybe, 10 or 20 years ago"* – P1, Enterprise Architect, Major City Council.

Knowledge is lost as developers retire, *"if somebody is approaching retirement and they are one of the only ones that know about that system, we should really start to have some sort of formal handover and transfer of knowledge as well. Otherwise, there is a knowledge gap and a skills gap there"* – P1, Enterprise Architect, Major City Council. This in turn creates risk, *"The big challenge is risk because when you get inside these things you don't always know what you're going to find, and there's a danger that if you do that, you break that"* – P10, CIO, High Street Retailer.

Knowledge loss combined with change in programming paradigms means new developers struggle to understand existing code bases, *"If you've only ever learned, for example, object-orientated languages, and then you're faced with a 20-year-old procedural language, it's completely different"* – P1, Enterprise Architect, Major City Council.

Developers tend to want to work on the latest technology, *"they're all [developers] chasing the next shiny thing the new languages and technology"* – P13, Head of SWE, Government Department. This inevitably leads to high developer turnover, this loss of knowledge makes future development hard, *"If their coding style, if they way they've written the application, is unfamiliar to the next group of people who come in, it's really hard. It takes longer to do development."* – P2, CTO of Start-up.

Finding developers with the correct skill-set and understanding of procedural code is a challenge, *"Typically, it was really difficult to get the programming skills in place to get enough of an understanding of all the nuts and bolts of the procedural code before we could migrate away onto any new solution "* - P1, Enterprise Architect, Major City Council.

However, P12 describes their experience with a skills gap, *"when they get old and retire you're not going to be able to find anybody who codes in that language"* - P12, CIO, Consultancy Company. They follow up with, *"well that's just not the case we found that we started to take in A-level apprenticeship scheme"*. Another participant echoes similar sentiment regarding the value of apprenticeships, *"We've got a number of apprenticeships. We've got a number of apprentices in our department as well. It's having that mix, to be perfectly honest."* - P1, Enterprise Architect, Major City Council.

The interviews reveal that there is a lack of knowledge in the languages that legacy systems were written in and the paradigms that were prominent at the time. Respondents describe how developers want to use the latest technology

and that many applications have evolved through significant shifts in technology. Additional, they reflect the world from many years ago, and participants describe how finding the skills to bridge this gap between old and new is difficult, and comes at a premium.

4.8 Complex System Architectures

The interviews reveal that system architectures become complex over time, P2 describes one situation *"Then you just get this layering of stuff that all, kind of, pretty much hangs together "* - P2 CTO of Start-up. This is echoed by P8, who described *"layers of sort of complexity sitting on the top of the base systems"* - P8, CIO, Large Enterprise. A lack of long term view causes systems to degrade, P2 describes how *"Each group that had come in had tried to create good modular well-structured code and all the rest of it. It was in 10 different languages and 30 different frameworks "* P2, CTO of Start-Up. They believe this is due to *"This is what happens when you've got project-based budgeting within an organisation, rather than product-based. "*, and that *" If you've got a product that you're continuously evolving, people care about the product and there's a much more long-term view taken with it. "* - P2, CTO Of Start-Up.

Participants describe to degradation over time as the systems as new functionality is added. There is described as being due to short term decisions being made when evolving legacy systems over time. Participants describe how this degradation leads to complex architectures, with systems that just "hang together" and are made up of many "layers".

5 Discussion

In this section discuss our findings. We also introduce a new type of technical debt, which we refer to as "Ecosystem Debt". We extend existing definitions of code related debt to include the impact of shifting technological paradigms, such as that which has occurred over the last few decades.

5.1 Practitioners Experience with Legacy Systems

Participants describe a mixed experience of legacy. They corroborate existing definitions within literature, including challenges with old code [4], and inflexibility [5]. We find similar responses to Khadka et al [16], in that participants view legacy systems as no longer support the organisations future direction. However, we note that our participants provided a more varied experience. Notably, we find a more pronounced conflict between participants from a technical background and those from management.

5.2 Legacy vs Technical Debt

Our participants experiences suggest that many of the challenges they face today are a consequence of the decisions (or lack of) made in the past. We therefore

draw parallels between legacy systems and that of technical debt [7] i.e decisions made in the past result in costs that need to be paid for in the future. It is therefore our view that legacy can be classified in terms of technical debt. Specifically, legacy software is an accumulation of technical debt that is never paid off. This can be compared to the stages of technical debt as described by Kruchten et al [19]. It is our view that Legacy software is software which has reached *tipping point* but never gone through any form of *remediation* phase.

Similar overlap between legacy and technical debt is highlighted in [15]. While we agree with the authors assessment that software practitioners managing technical debt could learn from the many decades of legacy modernisation research, we also believe the reverse to be true; practitioners managing legacy can borrow from research (including tools and techniques) on managing technical debt. Indeed, some of the benefits of this approach have already been highlighted by [14].

Many organisations are faced with a difficult choice; do they modernise or do they maintain? We believe that presenting legacy in technical debt terms goes some way to helping organisations make that decision. Technical Debt can give insight into the current state of a software system [18, 19], moreover it can provide a shared vocabulary between technical and non-technical practitioners, giving technical practitioners the tools needed to convey the risk and costs associated with maintaining an application [9, 2].

5.3 Ecosystem Debt

The size, scale and age of large scale legacy systems brings a unique set of symptoms that do not map directly to current types of technical debt [1, 19]. We note that practitioners describe a complex application landscape that often work together, typically supporting a number of services and staff. To the authors this is very similar to the definition of a software ecosystem as outlined in [20]. And while there has been some research into ecosystem health [11], as well as technical debt within the context of a software ecosystem [21] we do not believe the ecosystem itself has been described in terms of technical debt. To this end we define ecosystem debt as follows

"Software systems, and the systems they support, create a software ecosystem. Like any ecosystem, the impact on one aspect can influence others. If dependencies between software systems are not managed, future changes to the individual software systems that make up this ecosystem can be costly, if not impossible"

While current definitions of technical debt capture the state of individual applications, we believe they fail to capture the challenges that arise from the evolution of (or to) a software ecosystem. Ecosystem debt is therefore the consequence of decisions made during an ecosystems evolution (such as inheriting external software, integrating different, independently developed applications) that are done for expediency, but which later cause friction and hinder development.

5.4 The Impact of Older Programming Languages and Paradigms

A common theme from all participants is having to contend with older languages and programming paradigms. Languages such as COBOL are no longer widely taught which makes developer recruitment difficult, indeed, many participants cited lack of available skills as a motivation for wanting to move away from legacy systems. However, the internal quality of legacy systems not necessarily being of low quality they still incur the same penalties of poor quality software (for example, difficulty onboarding new developers, hard to understand code) [16]. We believe this to be due to the cognitive gap [25] which opens up as programming languages become dated. In this instance, new developers have to contend with learning not only COBOL, but also understanding the way COBOL applications were developed (structured, procedural code vs Object Oriented). This cognitive gap makes it harder to onboard new developers. This has a compounding affect as a lack of knowledge can itself cause technical debt as code quality is reduced.

5.5 Threats to Validity

External Validity: The organisations who were part of this study were all located within the United Kingdom, including Government Departments. These organisations have evolved within the UKs political and regulatory environment, as a consequence our findings may not be generalisable outside of the context of the UK. However, to mitigate against this, we interviewed a range of different organisations, including large scale enterprises.

Internal Validity: The target demographic for this study was practitioners involved in large scale legacy systems. This represents a very small, focused demographic within the IT and Software industry, as such a snowball sampling technique was used to identify participants. To limit potential bias from respondents in similar industries, we have included respondents from a varied industry background, including a mix of enterprise and government departments.

Conclusion Validity: For this paper we used semi-structured open ended interviews to collect data. These interviews were refined and revised after initial interviews were conducted. We utilised open ended questions, including allowing practitioners the opportunity discuss anything they felt relevant to limit the impact of the researchers own biases.

6 Conclusions

Organisations that manage legacy systems at scale face a set of unique challenges. They manage complex software landscapes that have evolved over decades. Over this period user expectations have evolved dramatically, as have technical paradigms. Current conceptual definitions of legacy systems gives little insight into the challenges associated with managing them, nor does it give much insight in how to avoid the transition from non-legacy to legacy.

In this paper we identify practitioner experiences while managing large scale legacy systems. We propose an alternative method for defining legacy systems by classifying respondents experiences in terms of types of technical debt. We present a new type of technical debt, which we refer to as "Ecosystem Debt" and extend current definitions of code related debt to include the effects of age, language choice and coding paradigm on code understanding.

We adopted a Grounded Theory approach to analysing 16 semi-structured, open ended interviews. We interviewed industry practitioners who maintain, or have maintained, large scale legacy systems. We used a snow ball sampling technique to identify industry practitioners for this paper. We selected industry practitioners from both large scale enterprises and large government department, including practitioners from both technical and non-technical backgrounds.

Our work in this paper contributes to understanding practitioner perspectives on Large Scale Legacy, to our knowledge there is only one similar study on this topic. Furthermore, we position Legacy in terms of technical debt, identifying similarities between both legacy and technical debt types. We additionally present a new type of technical debt, and expand current understanding of existing types of technical, namely code related debt.

7 Acknowledgements

We would like to acknowledge and thank the participants who took part in this study. Many of which are in senior positions, as such we appreciate them taking the time to participate in this study.

References

1. Alves, N.S., Mendes, T.S., de Mendonça, M.G., Spínola, R.O., Shull, F., Seaman, C.: Identification and management of technical debt: A systematic mapping study. Information and Software Technology **70**, 100 – 121 (2016)
2. Arvanitou, E.M., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Stamelos, I.: Monitoring technical debt in an industrial setting. In: Proceedings of the Evaluation and Assessment on Software Engineering. p. 123–132. EASE '19, Association for Computing Machinery, New York, NY, USA (2019)
3. Bass, J., Monaghan, B.: Legacy systems interview guide (Jul 2020). https://doi.org/10.17866/rd.salford.12662537.v1
4. Bennett, K.: Legacy systems: coping with success. IEEE Software **12**(1), 19–23 (1995)
5. Bisbal, J., Lawless, D., Wu, B., Grimson, J.: Legacy information systems: Issues and directions. IEEE Software **16**(5), 103–111 (1999)
6. Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., Zazworka, N.: Managing technical debt in software-reliant systems. pp. 47–52 (01 2010)
7. Cunningham, W.: The wycash portfolio management system. SIGPLAN OOPS Mess. **4**(2), 29–30 (Dec 1992)
8. Dedeke, A.: Improving legacy-system sustainability: A systematic approach. IT Professional **14**(1), 38–43 (2012)

9. Eisenberg, R.J.: A threshold based approach to technical debt. SIGSOFT Softw. Eng. Notes **37**(2), 1–6 (Apr 2012)
10. Fowler, M.: bliki: Technicaldebtquadrant (2020), https://martinfowler.com/bliki/TechnicalDebtQuadrant.html
11. García-Holgado, A., García-Peñalvo, F.J.: Mapping the systematic literature studies about software ecosystems. In: Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality. p. 910–918. TEEM'18, Association for Computing Machinery, New York, NY, USA (2018)
12. Gholami, M.F., Daneshgar, F., Beydoun, G., Rabhi, F.: Challenges in migrating legacy software systems to the cloud — an empirical study. Information Systems **67**, 100–113 (2017)
13. Glaser, B.G.: The discovery of grounded theory : strategies for qualitative research (2003)
14. Gupta, R.K., Manikreddy, P., Naik, S., Arya, K.: Pragmatic approach for managing technical debt in legacy software project. In: Proceedings of the 9th India Software Engineering Conference. p. 170–176. ISEC '16, Association for Computing Machinery, New York, NY, USA (2016)
15. Holvitie, J., Licorish, S.A., Martini, A., Leppänen, V.: Co-existence of the'technical debt'and'software legacy'concepts. In: QuASoQ/TDA@ APSEC. pp. 80–83 (2016)
16. Khadka, R., Batlajery, B., Saeidi, A., Jansen, S., Hage, J.: How do professionals perceive legacy systems and software modernization? pp. 36–47. No. 1, IEEE Computer Society (2014)
17. Khadka, R., Saeidi, A., Idu, A., Hage, J., Jansen, S.: Legacy to soa evolution: A systematic literature review. Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments pp. 40–70 (01 2012)
18. Kontsevoi, B., Soroka, E., Terekhov, S.: Tetra, as a set of techniques and tools for calculating technical debt principal and interest. In: Proceedings of the Second International Conference on Technical Debt. p. 64–65. TechDebt '19, IEEE Press (2019)
19. Kruchten, Philippe, a.: Managing technical debt : reducing friction in software develpment. SEI series in software engineering (2019)
20. Manikas, K., Hansen, K.M.: Software ecosystems–a systematic literature review. Journal of Systems and Software **86**(5), 1294–1306 (2013)
21. McGregor, J.D., Monteith, J.Y., Zhang, J.: Technical debt aggregation in ecosystems. In: Proceedings of the Third International Workshop on Managing Technical Debt. p. 27–30. MTD '12, IEEE Press (2012)
22. Razavian, M., Lago, P.: A systematic literature review on soa migration. Journal of Software: Evolution and Process **27**(5), 337–372 (2015)
23. Rios, N., de Mendonça Neto, M.G., Spínola, R.O.: A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. Information and Software Technology **102**, 117 – 145 (2018)
24. Wolff, E., Johann, S.: Technical debt. IEEE Software **32**(4), 94–c3 (2015)
25. Zaytsev, V.: Open challenges in incremental coverage of legacy software languages. In: Proceedings of the 3rd ACM SIGPLAN International Workshop on Programming Experience. p. 1–6. PX/17.2, Association for Computing Machinery, New York, NY, USA (2017)
26. Zazworka, N., Shaw, M., Shull, F., Seaman, C.: Investigating the impact of design debt on software quality. pp. 17–23 (2011)