Master’s Thesis – Master Sustainable Business and Innovation

Future proof ESG reporting software: How do architecture styles and usability relate to it?

Master thesis Sustainable Business and Innovation

Name	Fieke Dhondt
Student number	9677267
Date	July 7, 2023
Supervisor	Dr. M. M. H. Chappin
Second reader	Dr. Iryna Susha
Word count	19774

ABSTRACT

Introduction - Evolving regulations and business needs are shifting the reporting of environmental, social and governance (ESG) principles to software tools that can collect, measure, assess and audit ESG data. ESG reporting software should be flexible and robust to adapt to regulations and unforeseen events and thus should be future proof.
Theory - Future-proof software tools should handle changes with acceptable risk and without compromising the quality delivered. Past literature highlighted how future-proof software tools are primarily formed by the design feature architecture style. The extent to which ESG software tools were future-proof and which software design factors related to this had not been widely covered in the literature. Therefore, this research focused on how a system architecture style relates to the extent to which an ESG reporting software tool is future proof. More specifically, this study looks at the flexibility and robustness of nine ESG reporting software solutions in combination with design features. Besides that, the usability of these software tools has been assessed to identify how the concepts future-proof and usability are related.
Methods - A combination of semi-structured interviews and a structured part were conducted with software architects and system implementers for these nine tools. In addition, desk research was performed, and usability scores from an additional independent research platform were consulted.
Results - Overall, the future-proof scores of the participating software vendors were generally high. The findings further show no clear relation between architecture styles and the future-proof ratings of the participating ESG reporting tools. However, other design features such as the cloud service methods, coding methods, offering and ESG type seemed to relate to flexibility or robustness. Regarding usability, only two tools received a high score, and it seemed that high flexibility was related to usability in two ways. For most tools, high flexibility resulted in lower usability, but in other cases, high flexibility and future-proof scores were related to high usability.
Discussion/Conclusion – In some instances, the difference in results were related to the specific ESG context. In other instances, it related to additional features that possibly impacted future proof. Thus, the findings imply that the extent to which ESG reporting software tools are future proof is not solely related to architecture styles or usability. Instead, numerous other factors relate to future-proof ESG software scores.

Keywords: ESG reporting software, architecture, future proof, flexibility, robustness, usability

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BPM	Business Performance Management
CPM	Corporate Performance Management
CSR	Corporate Social Responsibility
CSRD	Corporate Social Responsibility Disclosure
EHS / EH&S	Environmental, Health and Safety
EPM	Enterprise Performance Management
ERP	Enterprise Resource Planning
ESG	Environmental Social Governance
GRC	Governance, Risk and Compliance
IaaS	Infrastructure as a Service
IS	Information System
LCA	Life Cycle Assessment
MSA	Microservice Architecture
PaaS	Platform as a Service
PMS	Performance Management System
SaaS	Software as a Service
S-ERP	Sustainable Enterprise Resource Planning
SOA	Service-oriented Architecture

1 INTRODUCTION

Environmental degradation is becoming a more significant problem, and stakeholders are demanding businesses to respond by integrating sustainability into their current practices (Fernandez-Feijoo, et al., 2014). As part of this response, corporate social responsibility (CSR) and managing environmental social and governance (ESG) issues are emerging as business propositions for long-term sustainability goals (Chelawat & Trivedi, 2016). CSR is merely a voluntary approach towards integrating social and environmental sustainability into the business processes and the interaction among stakeholders (Commission of the European Communities, 2001). In comparison, ESG integrates environmental, social, and governance into a corporation's business model. The difference between the concepts is that CSR approaches governance indirectly, whereas governance is explicitly incorporated in ESG (Gillan, et al., 2021).

One output of ESG and CSR is sustainability reports that intend to communicate the sustainability aims to shareholders and other stakeholders (Du, et al., 2010; Fortuna, et al., 2020). Even though there is an increase in reporting due to stakeholder pressure (Fernandez-Feijoo, et al., 2014; Almagtome, et al., 2020), sustainability reporting still contains a different quality and degree of transparency than traditional financial reporting (Littan, 2019). One way to improve the quality of reporting is through a mandatory reporting directive (Wang, et al., 2018; Mion & Loza Adaui, 2019). Recognising these challenges, the Corporate Social Responsibility Disclosure (CSRD) was accepted by the European Union (EU) in November 2022 as a mandatory reporting framework.

The CSRD has been developed to set one clear directive for disclosing social and environmental challenges and how companies operate and manage these issues. The framework is based on a double materiality framework, defined as the risk to the organisation from sustainability issues and the organisation's impact on people and the environment (European Commission, 2021a). The CSRD introduces eleven disclosure requirements that cover relevant ESG elements for corporations in the EU. Corporations are required to digitally tag more than 1000 sustainability data points to contribute to the single European access point (ESAP) and seize the opportunities digital tools offer. Eventually, the ESAP will be a digital platform that collects and stores corporations' publicly disclosed financial and sustainability information (European Commission, 2021b).

Mandatory ESG reporting, such as the CSRD, requires corporations to deliver reliable data and to measure, assess and monitor their impacts (McEwan, et al., 2021). Spreadsheets and other forms of manually tracking the impact of the organisations are inefficient and unable to grasp the completeness and validity of the data needed to report (Bakarich, et al., 2020). Therefore, business information system (IS) based ESG reporting software is becoming necessary to support ESG performance tracking and increase the quality of reporting to external stakeholders. Pan, et al. (2022) also acknowledge growing opportunities for intelligence business software solutions to increase visualisation, track, and benchmark an organisation's ESG performance. ISs are software and hardware networks that gather, store and process data intending to provide knowledge and digital products (Thambusamy & Salam, 2010; Loeser, et al., 2017; Helbig, et al., 2021).

Continuously changing regulations, and new requirements towards data delivery, such as the ESAP, require ESG reporting software to be future proof. Meaning that information systems should be flexible in responding to future laws, regulations, and operational, organisational, and consumer requirements (Said, et al., 2015). Pee et al. (2021) also recognise the need for flexible and resilient systems, as regulations and unprobeable events will change the future. The information system platforms should furthermore be extensible and incorporate mandatory reporting requirements to comply with current regulations and be able to uptake future requirements (Helbig, et al., 2021). Thus, a future proof software system should be flexible enough to adapt to change and be robust enough to handle changes.

When focusing on information systems in general, it seems that de degree to which software tools are future proof has not been studied extensively. It has been mentioned how specific design features of the software, and specifically the architecture style of a software tool, is an important aspect for future proof systems (Furrer, 2019). Furrer (2019) also identified how specific quality characteristics relate to future proof software systems. Another study in this context links flexibility and robustness to future proof software systems (Bass, et al., 2012), but the degree to which specific software tools are future proof has not been included. When looking more closely at flexibility and robustness, specific software flexibility characteristics have been compared in various literature (Knoll & Jarvenpaa, 1994; Mahinda & Whitworth, 2004; Gebauer & Schober, 2006; Chen, et al., 2009; Thuan, et al., 2020) and various robustness components have been discussed (Losavio, et al., 2003; Barber & Salido, 2015). Thus, it has been mentioned how architecture styles and quality characteristics might relate to future proof software and that these software tools should be flexible and robust to be future proof.

Besides the focus on future proof aspects, the usability of business IS in changing environments is an important aspect. Usability is one of the quality characteristics that have been mentioned by Furrer (2019). Business IS should not only be able to adapt to uncertain events but should also satisfy user requirements regarding the usability of the system (Palanisamy & Boyle, 2010). Various relations between flexibility and usability have been identified. Various studies examined the interplay between flexibility and usability and highlighted how flexibility had a positive (Palanisamy, 2012; Li & Nielsen, 2019) or negative (Lidwell, et al., 2010) impact on usability. Other studies have compared information systems on an aspect related to usability, but mainly focused on system adoption (Jamous, et al., 2012; Hoang, et al., 2019; Zhang, et al., 2021) and the future readiness of users (Mitropoulos & Douligeris, 2011; Green, et al., 2014). It seems that previous studies did not compare various information systems on usability in combination with their ability to be future proof but did identify various linkages between usability and flexibility.

Studies above specifically focused on general IS research; when looking at prior research on ESG software, literature covered the adoption of ISs and their link to sustainability reporting in Australian organisations (Seethamraju & Frost, 2016; Hoang, et al., 2019). Other studies examined the value and specific capabilities environmental enterprise systems (EES) deliver by comparing various software vendors (Hoang, et al., 2016; Jamous, et al., 2012) or through interviews with vendors and organisations (Hoang, et al., 2017). Even though the study from Seethramraju & Frost (2016) does suggest that future research could examine the ability of these systems to adapt in an organisation or technical context, it seems that the specific degree to which these systems are future proof has not been studied in the context of ESG.

Current studies on ESG-related business information systems have mainly focused on the adoption and value of these specific systems (Jamous, et al., 2012; Hoang, et al., 2016; Seethamraju & Frost, 2016; Hoang, et al., 2017; Hoang, et al., 2019). Even though it has been noted in various literature that the adaptability, resilience and flexibility of information systems are required to keep track of changing sustainability requirements (Said, et al., 2015; Helbig, et al., 2021; Pee, et al., 2021), this has not been widely covered. Thus, it seems that the extent to which current ESG reporting software tools are future proof has not been studied. In addition, the question of how the architecture exactly relates to future-proof systems has not been extensively studied, especially not with a focus on ESG reporting software. Lastly, the literature on the relation between flexibility in the context of future proof ESG reporting software and usability is limited. Thus, to what degree the current ESG reporting software systems are future-proof and how architecture and usability relate to this still need to be determined.

Therefore, this study aims to cover how a systems architecture style and usability relate to future proof ESG reporting software. Specifically, this research will explore the relation between architecture and future proof ESG reporting software and future proof ESG reporting software and usability by means of comparing existing ESG reporting software. Hence the following research question:

How are system architecture and usability related to ESG reporting software system's ability to be future proof?

1.1 Scientific and societal relevance

The contributions of this thesis relate to scientific, societal, and practical relevance. The contribution to current research is threefold and will be discussed first. Following, the societal and practical relevance will be discussed.

A rise in sustainability directives and regulations increases the need for ESG reporting tools that are future proof to respond to future changes. However, the actual degree to which ESG reporting tools are future proof has not been widely covered in the literature. Thus, this study enhances existing literature by linking future proof concepts to the specific context of ESG reporting software. More specifically, this research links the existing literature on future proof concepts, split in flexibility and robustness, to existing ESG reporting software. In addition, design features have been mentioned in past research as an important aspect of future proof software. The link between these software design features and future proof ESG reporting software has not been studied in a case study format. Therefore, this thesis will contribute to the existing literature by examining if there is a relation between design features and the extent to which ESG reporting software tools are future proof. Lastly, usability has been identified as an important attribute for software systems, and a possible link between future proof software and usability has been identified. This thesis will add to the existing literature by examining the possible link between the extent to which ESG reporting software tools are future proof and usability.

Next to the scientific relevance, the findings of this study hold societal relevance as well. This relevance mainly relates to increased knowledge of ESG reporting software and the importance of its design. ESG reporting software tools are important for visualisation, tracking and benchmarking the performance of organisations. Data transparency and collection are an important part of ESG, especially in the context of new upcoming regulations such as the CSRD. By addressing specific design features of these tools, the findings can contribute to the development and implementation of software solutions that promote transparency, improved tracking and benchmarking of ESG data.

In addition to the scientific relevance, the results of this thesis will also relate to practical relevance for software vendors, system implementers and end-users. This thesis generates insights for vendors in how ESG reporting software can be designed to be future proof. Thus, this study can possibly guide specific aspects of the development of ESG software solutions that are flexible and robust for future changes and upcoming regulations. The research is also relevant for implementers and end-users. When implementers and end-users are more aware of possible design features related to future proof software, it allows them to consider and incorporate these features more effectively and in accordance with their own objectives.

2 THEORETICAL BACKGROUND

2.1 Business information systems

Business information systems can be subdivided into various types of systems, such as enterprise performance management (EPM) systems, enterprise resource planning (ERP) systems, governance risk and compliance (GRC) systems, environmental, health and safety (EHS) software and environmental, social and governance (ESG) reporting systems, as presented in Figure 1. Each of these systems has a different role to play in an organisation, which will be elaborated upon below.

The first system type, EPM systems, are used to manage, analyse and report on data and complement ERP systems. EPM cloud solutions should align with ERP systems to ensure agility. EPM systems are also referred to as corporate performance management (CPM) systems, business performance management (BPM) systems, or performance management systems (PMS) (Druzhaev, et al., 2019). EPM systems are not necessarily ESG reporting software but can contain an ESG reporting solution.

Whereas EPM systems are focused on the business process, ERP systems are designed to manage and process data from different departments, such as HR, accounting, or finance, in one system. As part of ERP systems, Sustainable Enterprise Resource Planning (S-ERP) systems are emerging (Chofreh, et al., 2014). S-ERP systems manage and report ESG impacts and align organisational operations, people, and products (Chofreh, et al., 2020). Some ERP vendors include an ESG reporting solution in their software systems, but this is not always the case.

Another system that can be used to track and organise organisational information is a GRC system. This type of IS system collects and stores GRC data in one tool to manage risks, guide organisations to comply with regulations and sustain a governance structure (Papazafeiropoulou & Spanaki, 2016). New opportunities are emerging, and vendors are integrating ESG risks and reporting add-ons into GRC tools.

In contrast to GRC systems, the EHS system focuses more specifically on environmental, health and safety risks and compliance. Organisations can use EHS software to capture and analyse EHS-related incidents. Information in these systems is often related to health and safety, waste management and sustainability. As a part of these software types already includes a sustainable component, it often includes an ESG solution (Hoang, et al., 2016; Verdantix, 2023).

ESG reporting solutions can be part of ERP, EPM, GRC, EHS or other systems, but ESG reporting software can also be a standalone software system. ESG reporting systems are software used by businesses to track, measure, manage and report on ESG principles. The software is specifically designed for ESG use and often complies with global ESG standards such as the CSRD (Enghaug & Hallan, 2022). Therefore, this research will include five types of business information systems: ERP, EPM, GRC, and EHS systems with an ESG solution and ESG systems.

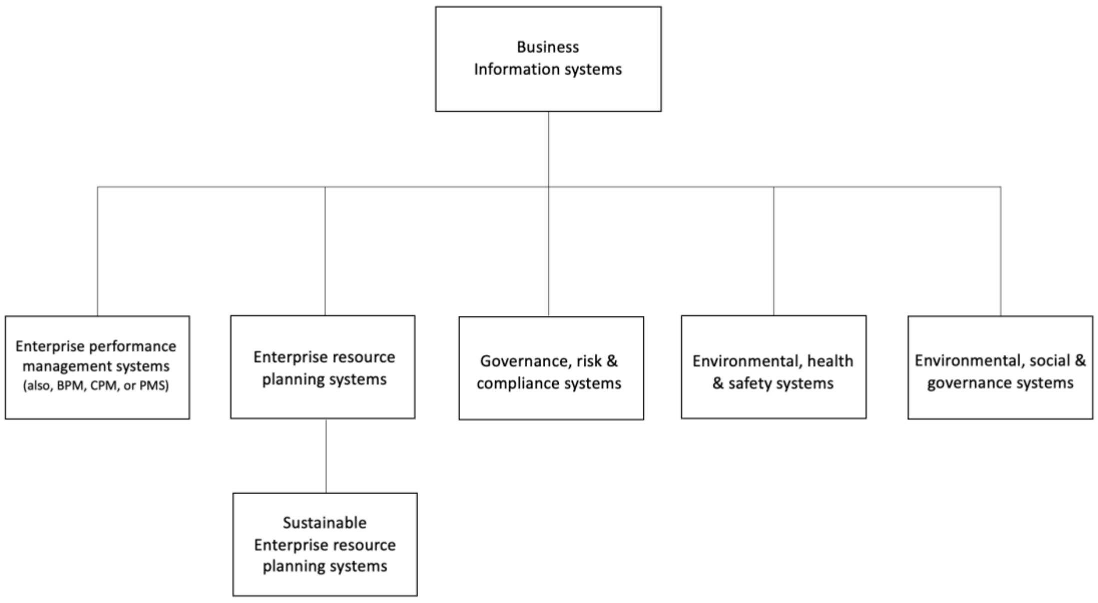

Business Information systems
Enterprise performance management systems (also, BPM, CPM, or PMS)
Enterprise resource planning systems
Governance, risk & compliance systems
Environmental, health & safety systems
Environmental, social & governance systems
Sustainable Enterprise resource planning systems

Figure 1 Types of business information systems (own figure)

2.2 Future ready and future proof

ESG software solutions must be able to cope with high uncertainty and be able to adjust to new requests and upcoming regulations (Said, et al., 2015; Helbig, et al., 2021; Pee, et al., 2021). Various definitions of future ready and future proof business information systems will be provided.

Future readiness of information systems can be understood as delivering continuous improvement and preparing for future challenges (Martinsons, et al., 1999). Future readiness relates to the learning and growth trajectory of an information system. Until now, future readiness was perceived mainly from an adopter approach and the degree to which a corporation is ready for information system products and services. Future readiness, in this sense, was thus primarily related to the internal use of information systems by corporations (Martinsons, et al., 1999).

A concept more related to the software system itself is the concept of future proof. Future proof is often linked to a technology that can handle complex change and unpredictability with low effort but with acceptable risk (Furrer, 2019). Therefore, the definition of future proof suits this thesis's aim better as it focuses more specifically on the ability of a software system to handle change without compromising the system's quality. Drawing on the existing information systems literature, there are two attributes of which future proof is composed. These are the flexibility and robustness of the business IS (Bauer & Maurer, 2011) and will be further explained in the following sections.

2.2.1 Flexibility

Flexibility has been used interchangeably with many related terms, such as customisability, adaptability and changeability. Definitions of flexibility slightly differ. Reichert and Weber (2012) define flexibility as the ability of a system to adjust to unpredictable inputs. Furrer (2019) refers to changeability and defines this as the adaptation of the system when a change occurs and the effort it takes. Thus, the first definition concerns the ability to handle a change, whereas the second definition relates to the ease of making a change. The second definition will be applied to this research as it is most suitable for the concept of future proof.

Flexibility is an important aspect of ESG software as regulations and legislations are constantly adjusted, and therefore systems need to be able to respond to new regulation requests (Cruz & Matos, 2023). Besides that, user needs and intrinsic motivations to become more sustainable are increasing the demand for new sustainability features in existing solutions. Software solutions should be able to respond to this in a timely matter (Hilpert, et al., 2014). There are two types of change related to ESG reporting flexibility: foreseen and unforeseen. Foreseen change is often related to changes that a business IS can prepare for; an example of this are regulations. Unforeseen changes can be linked to a change in user demands, which, especially in competitive markets, can vary rapidly (Jacome, et al., 2011).

Information systems literature has widely studied flexibility (Gebauer & Schober, 2006; Chen, et al., 2009; Thuan, et al., 2020). However, the approach to identifying the flexibility of a system differs. Two studies that mainly focused on the system's ability instead of the user's perspective were carried out by Knoll and Jarvenpaa (1994) and Mahina and Whitworh (2004). Knoll and Jarvenpaa (1994) divide flexibility in terms of functionality, use and modification. Mahina and Whitworh (2004) conceptualised the general flexibility of the system, the flexibility by detection and the flexibility by the response. Flexibility in modification and flexibility in response are most suitable to measure changeability as the attributes relate to the ease with which it can be modified. Certain components are left out as they do not relate to the definition of flexibility used in this study. Table 1 presents the combination of flexibility attributes.

Table 1 **Flexibility attributes of business information systems**

Main concepts	Components	Conceptual definitions	Adapted from
Flexibility in modification	Goal adjusting	The ability to change the system according to feedback.	(Knoll & Jarvenpaa, 1994; Hilpert, et al., 2014)
Flexibility in response	Responsiveness	The speed with which the system can be adjusted.	(Knoll & Jarvenpaa, 1994; Hilpert, et al., 2014)
	Adaptation	The ease in adjusting the software to a new end-use which has not been recognised before.	(Mahinda & Whitworth, 2004)

2.2.2 Robustness

The system's robustness relates to how the system can withstand uncertainty till a certain amount. A robust system should withstand unpredictable changes and contain its functionality (Knoll & Jarvenpaa, 1994; Jiménez-Ramírez, et al., 2015). In terms of ESG, as regulations and business needs are continuously changing (Cruz & Matos, 2023), a software system should be able to handle these foreseen and unforeseen changes (Jacome, et al., 2011).

There are three main components of robustness: stability, recoverability and reliability. Recoverability can be identified according to the system's ability to recover from errors or failures, and reliability as the ability to continue performing in unpredictable conditions (Barber & Salido, 2015). The stability of a system has often been used interchangeably with robustness, but the definition for both slightly differs. Barbey & Salido define stability (2015) as a system that can adapt to a new valid solution with slight modifications. However, this definition of stability overlaps with flexibility. Another definition of stability is the ability of the system to prevent unintended consequences from modifications made (Losavio, et al., 2003). The definition for stability of Losavio, et al. (2003) will be used for this study in combination with the definitions of recoverability and reliability by Barber & Salido (2015) to form the concept of robustness. The three robustness components are presented in Table 2.

Table 2 Robustness components of business information systems

Robustness components	Definition	Adapted from
Stability	The ability of the system to prevent unintended consequences from modifications made.	(Losavio, et al., 2003)
Recoverability	The system's ability to recover once an error or failure occurs.	(Barber & Salido, 2015)
Reliability	The capability of the software to continue performing under unexpected conditions.	

2.3 Software design features

The design of the software has been identified as a possible factor that could relate to the extent to which a software tool is future proof. One feature that has been mentioned as a factor that could influence future proof software systems is the architecture of a software system (Bass, et al., 2012; Furrer, 2019).

2.3.1 System architecture

As defined by ISO, IEC, and IEEE (2022), the architecture of a system is the concepts and properties essential for the software and the leading principles that realise the evolvement over time. Software architecture continuously changes and adapts to new requirements (Oudshoorn, 2004; Furrer, 2019). These architectures are structured according to a horizontal and vertical layer system layer, as presented in Figure 2. The horizontal layer provides containers for each function. There are five horizontal layers: business architecture, applications architecture, information architecture, integration architecture and technical architecture. The horizontal layers are mainly focused on the functionality of an organisation, whereas the vertical layer provides additional quality components such as safety, security and real-time architecture. Both the horizontal and vertical layers cross each other as each vertical component is also relevant to the horizontal components (Furrer, 2019).

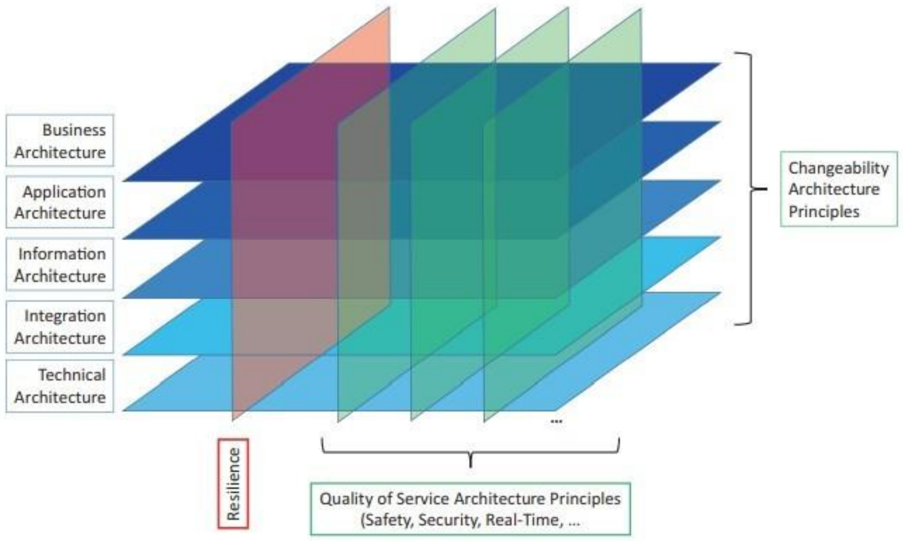


Figure 2 Typology of architecture principles (Furrer, 2019)

Quality components are part of the vertical architecture and are important requirements for the architecture. There are various software qualities, depending on the end use of the system. The quality characteristics of sustainability software have been studied in literature and encompass many characteristics that can be assigned (Koziolek, 2011; Calero, et al., 2013; Calero, et al., 2014; Koçak, et al., 2014; Venters, et al., 2014; Saher, et al., 2020). Quality characteristics that these studies have identified are maintainability, portability, usability, efficiency, compatibility, and reliability. Each quality characteristic can be further divided into sub-characteristics to enhance the measurability of these concepts (see Figure 3). These sub-characteristics are, in some instances, linked to the architectural styles due to the design of the architecture, whereas others, such as efficiency, reliability and usability, are overarching characteristics. Efficiency will not be considered a quality characteristic in this study as it contains trade-offs with the attribute flexibility (Subramanyam, et al., 2012). Reliability will not be used as it is too similar to robustness. Usability will be explained later as a separate overarching quality characteristic.

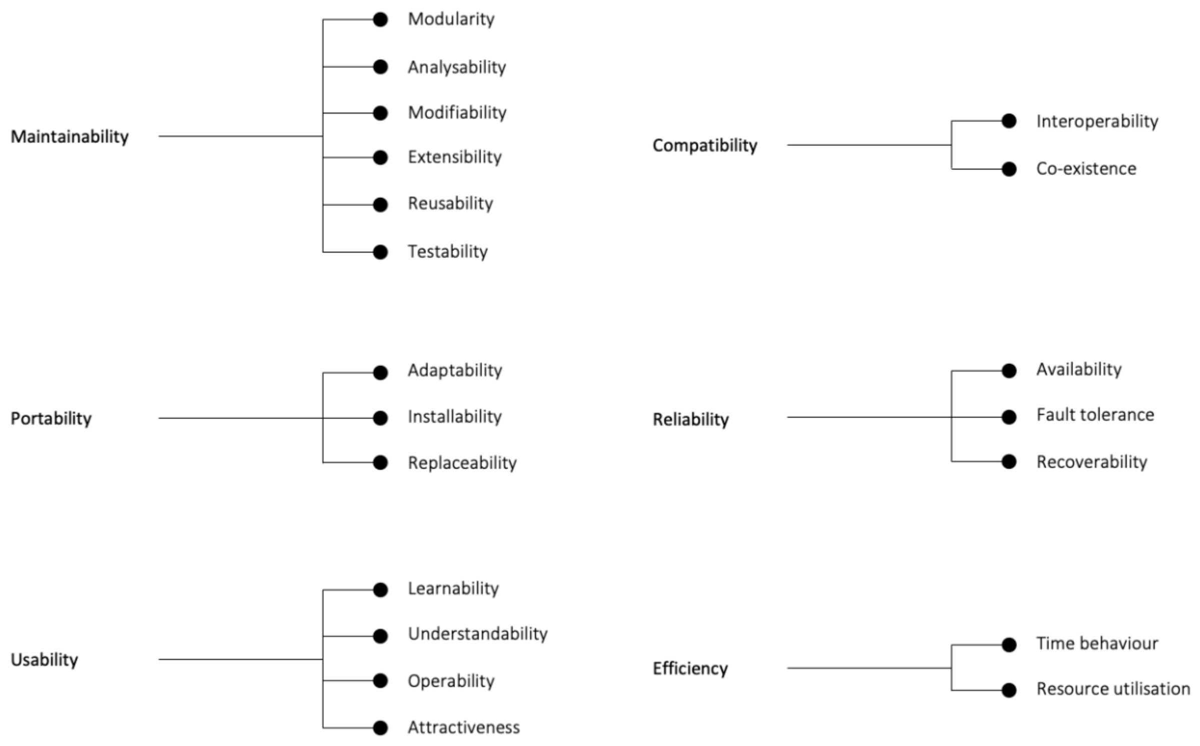


Figure 3 Quality characteristics and quality sub-characteristics as adapted from (ISO, 2011; Saher, et al., 2020)

Various styles of architecture are suitable for systems that include ESG reporting. Some common architectures are monolithic, service-oriented, microservice and serverless architecture; Figure 4 presents an overview of these styles. A software tool is not necessarily limited to one architecture and can use a combination of architectures. The various architectural styles and corresponding sub-characteristics have been linked and will be explained below, and a summary is presented in Table 3.

The first style of architecture, monolithic architecture, is a single unit that holds all the components in one system, such as the collection of data, storage and user interface. This relatively simple architecture is difficult to scale (Ponce, et al., 2019). Monolithic architectures are the least related to the system quality characteristics and only correspond with reusability when there is a small code file (Slamaa, et al., 2021). Overall, the monolithic architecture is expected to relate the least to flexibility due to its lack of scalability (Götz, et al., 2018). Consequently, as it is not related to flexibility, it is unlikely that this architecture style relates to future proof software systems. A combination of a monolithic architecture and cloud deployment will likely lead to a more future-proof system as the installability and replaceability of the cloud increases flexibility and scalability (Fink & Neumann, 2009; Lenhard, et al., 2013). However, it is still unlikely that a monolithic architecture style will relate to future proof systems.

Service-oriented architectures contain multiple services that can be reused for multiple applications. This architectural style is generally perceived to relate to flexibility and robustness (Tsai, et al., 2014; Hustad & Olsen, 2021). Service-oriented architecture has also been linked to multiple quality characteristics such as reusability, analysability, modularity and replaceability (Haoues, et al., 2017; Slamaa, et al., 2021). The only limitation is that a service-oriented architecture binds all services to one context, which could limit flexibility (Cerny, et al., 2018). Nevertheless, a service-oriented architecture is expected to relate to a future proof software system.

The third architecture style, microservice architecture, is an improved variation of the service-oriented architecture (Sewak & Singh, 2018) and comprises connected independent modules. These parts all have their own infrastructure and databases. Microservice architectures are likely to relate to flexibility and robustness, as they are more scalable and reliable than other architectures (Hasselbring & Steinacker, 2017; Slamaa, et al., 2021). Microservice architectures are linked to the quality characteristics of modularity, replaceability, reusability and analysability (Newman, 2015; Balalaie, et al., 2016; Auer, et al., 2021; Slamaa, et al., 2021). Thus, a microservice architecture is expected to relate to future proof software systems.

The last architecture style is the serverless architecture relating to software tools hosted on external cloud providers such as Microsoft Azure or AWS. This type of architecture is expected to be more efficient than other architectures (Rajan, 2018). Quality characteristics that are linked to a serverless architecture are analysability, modularity and interoperability (Racicot, et al., 2019; Poth, et al., 2020). Besides, this architecture is expected to relate to robustness and high flexibility (O'Meara & Lennon, 2020; Poth, et al., 2020; Lakhai & Bachynskyy, 2021). Thus, a serverless architecture is expected to relate to future proof software systems.

Proposition 1: Serverless, Microservice, service-oriented architecture styles will likely relate to future proof software systems, whereas monolithic architecture will likely not relate to future proof software systems.

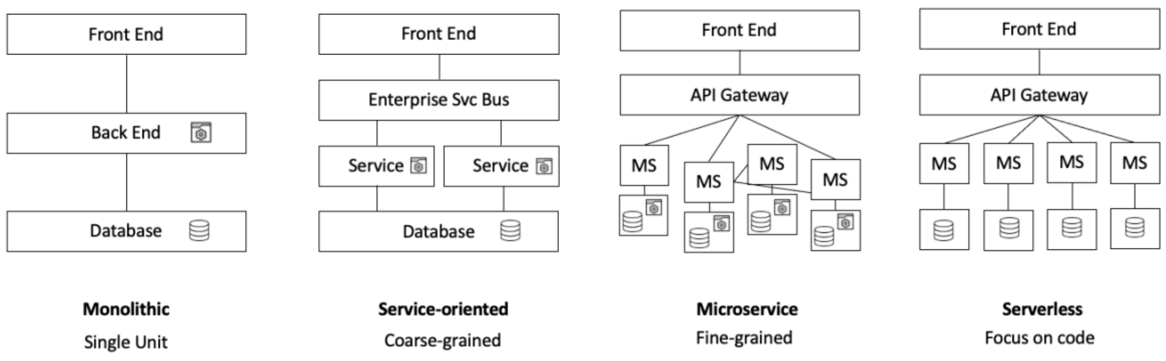


Figure 4 Simplistic representation of architecture styles as adapted from Taibi et al. (2020)

In addition to the architecture style, the deployment model has been linked to various quality characteristics. A software's deployment model influences how end-users can access software. There are two common deployment models for software: on-premises and cloud. Software tools that are on-premises are installed on the client's servers, whereas the cloud can often be reached through a web browser. A combination of the deployment model and the architecture style leads to different quality characteristics, depending on the deployment model. Cloud deployment models are flexible and scalable and can be accessed from any location. The specific deployment model can also influence quality sub-characteristics such as reusability, replaceability, interoperability and installability of the system. These quality characteristics are generally present for cloud deployment models (Garg, et al., 2011; Oh, et al., 2011; Muntes Mulero, et al., 2013). The quality characteristics of cloud deployment models will be combined with the other four architecture types as this style is often combined with other architectures (Al-Debagy & Martinek, 2018).

Table 3 Overview of system architecture and corresponding quality sub-characteristics

Quality	Architecture						
Sub-characteristics	On-premises Monolithic	Cloud Monolithic	On-premises Service-oriented	Cloud Service-oriented	On-premises Microservice	Cloud Microservice	Cloud Serverless
Analysability			x	x	x	x	x
Installability		x		x		x	x
Modularity			x	x	x	x	x
Interoperability		x		x		x	x
Replaceability		x	x	x	x	x	x
Reusability	x	x	x	x	x	x	

2.3.1.2 Usability

The usability of a system is context-specific (Speicher, 2015) and depends on the software systems design and workload (Poth, et al., 2020). It is an overarching characteristic that is relevant for all architecture types. Therefore, this study will consider usability a separate quality characteristic. Usability of the system refers to how the system can be used, how easy it is to learn the system, and the operability and attractiveness of the system. Operability is the capacity and ease to control the system. Other characteristics, such as understandability and learnability, relate to the experiences of previous users in learning and understanding the system (Saher, et al., 2020).

Past literature has identified various types of relations between flexibility and usability. According to Lidwell et al. (2010), there is a trade-off between flexibility and usability, meaning that high flexibility would decrease usability. This trade-off is explained through the complexity of a software system. The complexity of a system increases with higher flexibility and, therefore, impacts the system's usability (Lidwell, et al., 2010). Nevertheless, this might not be the case for all software, as some enterprise software systems are considered to be both highly flexible and usable (Dvořák, et al., 2017). Other studies did find a link between inflexible software systems and usability problems (Mahrin, et al., 2008; Li & Nielsen, 2019; Rakovic, et al., 2020). Rakovic, et al. (2020) analysed various software systems from the user's point of view through a survey. Inflexible ERP software systems gained lower scores from users (Rakovic, et al., 2020). Flexible software, on the other hand, leads to user-friendly systems and helps prepare organisations to face changes (Palanisamy, 2012; Li & Nielsen, 2019).

Proposition 2: A low flexibility rating and, thus, a low future-proof rating will likely relate to low usability.

3 METHODOLOGICAL FRAMEWORK

This section covers the methodology used for this thesis and explains the research design, sampling strategy, data collection, operationalisation, data analysis, and research quality indicators.

3.1 Research design

The aim of this thesis is to understand how various software architecture types and usability relate to various extents of future proof ESG reporting systems. This thesis followed a comparative case study research design. Therefore, various ESG reporting software tools were compared to identify patterns. According to Bryman (2012), this design type compares various cases to differentiate characteristics and eventually identify patterns. A deductive approach is used as the characteristics were derived from existing literature. A qualitative approach suited this thesis most, as the thesis aimed to gain more in-depth knowledge about how software architecture and quality characteristics relate to future proof ESG software. The data was analysed at a single point in time. The unit of analysis of this thesis was defined as ESG reporting software tools for the collection, measurement and reporting of ESG performance for enterprises.

3.2 Sampling strategy

Software vendors were selected according to a generic purposive sampling approach (Bryman, 2012). The first step of sampling was done by selecting vendors through the research platforms Verdantix and IDC. Verdantix is a research platform and advisory firm that specialises in, among others, digital strategies for ESG & sustainability (Verdantix, 2022). The International Data Corporation (IDC) is a market intelligence provider for information technology and includes an ESG module (IDC, 2023). Both firms provide market information about ESG vendors. The vendors were selected according to their link with ESG reporting. A total of 56 vendors were identified accordingly.

The second step of selecting vendors was performed by identifying if they included an ESG reporting module that contained environmental, social and governance aspects. As some of the vendors did not include a reporting module on all ESG principles, the population size was reduced to 49 vendors. Due to limitations, only 25 of these 49 vendors were contacted to participate in an interview. Eventually, nine software tools participated in this study after reaching out to all 25 vendors.

Larger software tools are usually implemented by consultants in a client's organisation. System implementers inform clients on how to adopt and integrate a software solution into their business operations. These system implementers were interviewed for each software solution to reduce the possibility of software vendors overestimating their tools. This thesis is written as part of an internship at a consulting firm. Therefore, a snowball sampling approach was used to sample the system implementers. Snowball sampling is defined by Bryman (2012) as a sampling approach where the researcher reaches out to a group of people and gathers other contacts through this group of people.

3.3 Data collection

The thesis was carried out according to three main phases. The first phase involved verifying the architecture styles, interview subjects and scales with industry experts. In the second phase, the system type, architecture styles and other relevant features were identified through desk research. In the third and last phase, semi-structured interviews accompanied by a structured part were held with representatives of software vendors and system implementers. Data on the software systems were collected through three phases, as presented in Table 4.

Table 4 Data collection and aim

Data collection phases		Aim
Preparation phase	Consultation with industry experts	Before the interviews were held, industry experts were consulted to provide feedback on the interview questions and the aim of the study.
Desk research phase	Documents and desk research	The aim of this phase was to get a general overview of the features and architecture of the vendor's system and to familiarise with the software before the interviews.
	External independent research firm	Usability scores were derived from an external independent research firm for six out of nine software to strengthen the findings of the research.
Interview phase	Test interview vendor	Before the interview with vendors were held, a test interview with a vendor was conducted to practice and to check the relevance of the interview questions.
	Semi-structured interviews vendors	This phase was used to validate the findings from the document and desk research and to get an understanding of the flexibility and robustness of the system.
	Semi-structured interviews implementers	The aim of the last data collection step was to reduce the overestimation of software vendors and increase validity by means of verifying the data. Furthermore, usability questions were included in these interviews to estimate the usability of the software.

Two industry experts reviewed interview questions and the aim of the thesis during the preparation phase. Furthermore, a test interview was held with an additional vendor to ensure that the question came across as intended. After this phase, several adjustments were made to the questions.

After the consultation with industry experts, desk research was carried out to familiarise with the software vendors and the solutions they provide. Online research was conducted through the vendor's websites, by attending webinars, watching YouTube videos from the vendors, community platforms, and an external independent research firm, as presented in Table 5 and Appendix A. Where possible, the usability findings of an external independent research firm were used to provide additional insights into the usability of the software.

The combination of a semi-structured interview with a structured part was used as certain questions required additional explanations, and a survey would not generate the required in-depth answers. However, to generate a score for flexibility, robustness and usability, Likert scale questions were implemented in the interviews. The interviews were conducted via Microsoft Teams, as most software vendors are based worldwide. The interview guide can be found in Appendix B.

All characteristics and future proof indicators were covered during these semi-structured interviews with representatives of the software vendors. Representatives of the software vendors were either software engineers or technical architects with knowledge of the system's architecture. The vendors were contacted either through representatives of the internship firm or directly by e-mail. In cases where the direct e-mail of software architects was known, these were contacted directly. In other instances, the general company e-mail was used. Next to that, for certain vendors, system implementers were interviewed through semi-structured interviews to validate the data from software vendors.

Eventually, all interviewed implementers came from within the internship firm. During the interviews with software vendors, contact details of possible implementers were requested. Only one vendor architect provided the contact details of an implementer, but even this implementer worked at the internship firm. In addition, five other consulting firms with an alliance or partnership with the interviewed vendors were contacted, but none responded. Clients that use these specific software tools were not contacted as they only experienced their specific case with the software tool, whereas implementers generally work for multiple clients and can therefore provide a more holistic view.

Fifteen interviews were held for nine software solutions to determine how various software architectures and system quality characteristics relate to the flexibility and robustness of ESG reporting systems. Eight vendors and seven implementers participated in the interviews. One implementer had worked with multiple software and therefore provided information for two software. The breakdown of people reached out to is provided as follows. A total of 107 people were reached out to, of which 76 were contacted through a direct e-mail, 21 through an info e-mail, seven through LinkedIn, and three through the vendor's website. Initially, 83 people were contacted, of which 52 did not respond after a reminder, 14 directed me further, 11 could not help me further, and six participated in the interviews. Seven additional interviews came out of the e-mails that were directed further, and two out of other interviews (see Figure 5).

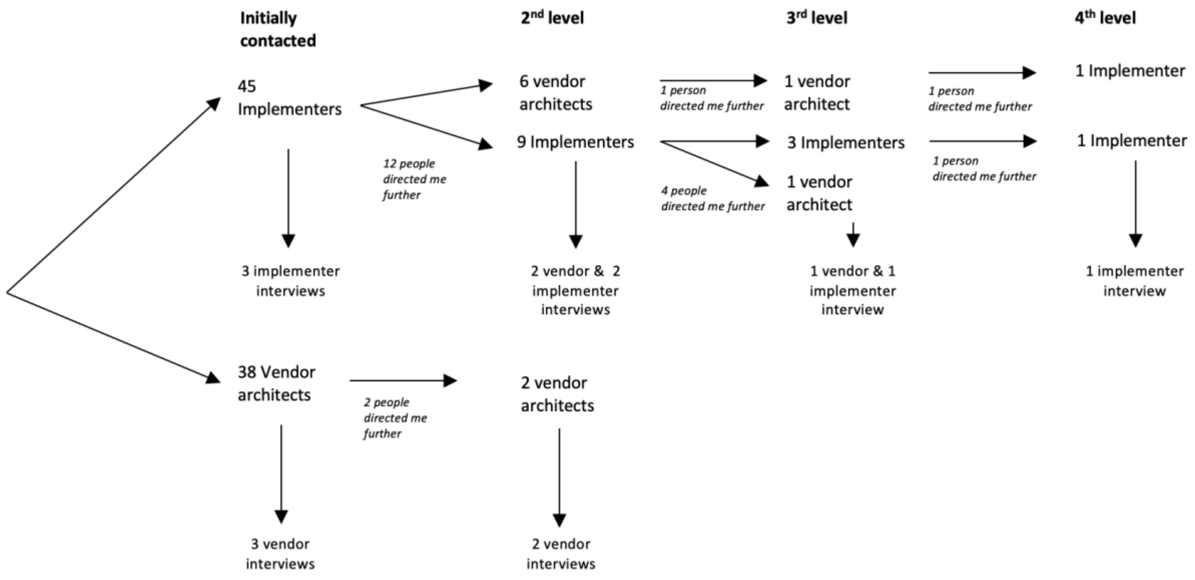


Figure 5 Overview of the number of people reached out to and eventual interviews.

An overview of the used data sources for each software is provided below in Table 5. For two software tools, the vendors did not respond or could not participate due to a conflict of interest. Therefore, a system implementer with more technical knowledge was interviewed. A system implementer is missing for four out of nine software tools. To fill these data gaps, insights into usability from an external independent research platform were used.

Table 5 Data sources

Name	Desk Research					Interviews	
	Website	Webinar	YouTube	Community platform	External research platform	# Vendor representatives	# Implementers
Software 1	✓	✓	✓	✓		1	1
Software 2	✓	✓		✓	✓	1	2
Software 3	✓	✓	✓		✓		2
Software 4	✓		✓		✓	1	
Software 5	✓				✓	1	
Software 6	✓		✓		✓		2
Software 7	✓					2	1
Software 8	✓	✓	✓		✓	1	
Software 9	✓		✓			1	

3.4 Operationalisation

Variables were operationalised according to the literature that is presented in the theoretical framework.

3.4.1 Future proof

The concept of future proof is rated on a high, medium, and low scale and combines the total scores from flexibility and robustness. A software tool is considered to be future proof if both flexibility and robustness are high, as presented in Table 6. The scaling processes for flexibility and robustness will be further explained in the next sections.

Table 6 Example of scoring future proof

	Flexibility	Robustness	Future proof
Software X	High	Medium	Medium
Software Y	High	High	High
Software Z	Medium	High	Medium

3.4.1.1 Flexibility

Two main concepts of flexibility were used in this study: flexibility in modification and flexibility in response. These concepts were split into three main components: goal adjusting, just-in-time adjusting and adaptation.

As had been mentioned earlier, one of the most important aspects of ESG reporting software tools is being flexible in responding to new sustainability standards and regulations and responding to new business needs of clients (Hilpert, et al., 2014). Therefore, the indicators defined by Knoll and Jarvenpaa (1994) and Mahina and Whitworh (2004) have been adjusted to fit the specific flexibility needs of ESG reporting software tools.

A five-point Likert scale has been used to scale the concept of flexibility in combination with insights from the interviews. The Likert scale range has been defined and adjusted together with two industry experts. The scale for flexibility is presented in Table 7.

Table 7 Scale of flexibility

Concept	Indicator	Range				
Goal adjusting	When a new sustainability regulation is implemented, the system can adjust easily	1 strongly disagree	2	3 neutral	4	5 strongly agree
Responsiveness	How long does it typically take to implement a new sustainability regulation?	1 > 8 weeks	2 7,8	3 4-6 weeks	4 2,3	5 1 week
Goal adjusting	When a new business need is recognised, the system can adjust easily.	1 strongly disagree	2	3 neutral	4	5 strongly agree
Responsiveness	How long does it typically take to implement a new functionality following a new business need?	1 > 1 year	2 10-12	3 7-9 months	4 4-6	5 1-3 months
Adaptation	Are new functionalities added for all users or separately for specific users	1 all				5 separately

Each participant was asked to explain the various concepts for the specific tool and was asked to rate the concept from 1 to 5, as explained in the interview guide in Appendix B. The scores for flexibility were noted for each question and then merged to a total score for that participant, as presented in Table 8. Eventually, all the scales were transformed to a high, medium or low score. The scoring matrix of the Likert scale for high, medium, and low scores is presented in Table 9. As multiple sources rated the software, these eventual low, medium and high scores were combined to gain a total score for a concept. These scores were used to get an initial impression of the flexibility of the software, and the reasoning behind these scores was further explained during the interviews.

In addition, the explanations given during the interviews were compared to see if vendors and implementers had a corresponding experience with the software tool and if the scores corresponded with the explanations given. If an interviewee would only give high scores (agree or strongly agree) but the explanations provided indicate that the software is not as flexible as the other participating software tools, the researcher would interfere and adjust the total given score for a software tool. This interference only occurred once for the concept robustness and will be explained later on.

Table 8 Example of scoring flexibility

Sources	Flexibility					
	x	y	Z	*Average*	*Score*	*Total*
Vendor Software x	3	4	5	*4*	*High*	*High*
Implementer Software x	5	4	5	*4,7*	*High*	

Table 9 Score rating according to scale

Scale rating	Score
1-2	Low
2,1-3,9	Medium
4-5	High

3.4.1.2 Robustness

The robustness concepts of Losavio et al. (2003) and Barber & Salido (2015) have been adapted to fit the robustness of ESG software tools. These software tools should be able to withstand adaptations made to the software without compromising the quality that is delivered.

For each concept, two to three questions have been formed to scale the robustness of the software tools, as presented in Table 10. Software reliability measures how well respondents think a software tool provides the required services (Roca, 2019). This concept has been adjusted to fit the requirements of changing ESG software tools. A possible way to identify the stability of a software tool is by looking at the unexpected effects after a change, such as bugs or downtime (Losavio, et al., 2003; Barber & Salido, 2015; Salama, et al., 2019). The statements focused on bugs have been specified to fit the ESG context to mainly focus on a change in the software due to a new regulation or reporting format and to fit the user's business needs. Recoverability can be measured in the amount of time it takes to recover and the degree to which the software is able to recover (Pan & Hu, 2014). The range for the recoverability time has been defined in accordance with industry experts and during the test interview.

The same scoring method applies as for flexibility (see Table 8 & Table 9). The insights from the interviews, Likert scale questions, and various sources are thus combined to gain a total score for the concept robustness and understand the reasoning behind it.

Table 10 Scale of robustness

Concept	Indicator	Range				
Reliability	How confident are you after changing the tool about the system's - Accuracy	1 not at all	2	3 neutral	4	5 extremely confident
Reliability	How confident are you after changing the tool about the system's - Reliability	1 not at all	2	3 neutral	4	5 extremely confident

Concept	Indicator	Range				
Stability	Unexpected bugs often occur after a change in the system - For a new regulation or reporting format	1 strongly agree	2	3 neutral	4	5 strongly disagree
Stability	Unexpected bugs often occur after a change in the system - To streamline to the user's business needs	1 strongly agree	2	3 neutral	4	5 strongly disagree
Stability	The system rarely experiences downtime or outages after a change is implemented	1 strongly disagree	2	3 neutral	4	5 strongly agree
Recoverability	The system can recover from failure or outages - Effective	1 strongly disagree	2	3 neutral	4	5 strongly agree
Recoverability	How long does it typically take to resolve system issues or bugs following a change in the system?	1 > a day	2 9-23 hours	3 6-9 hours	4 1-6 hours	5 < 1 hour

The interviews were crucial in verifying the alignment between the assigned scores and the accompanying explanations. In one case, there was a discrepancy between the given scores for robustness and the interview insights. The implementer explained during the interview that this specific ESG software was considered to be highly robust, but the total score for robustness came down to medium. The explanation and eventual robustness score thus differentiated. In addition, another implementer that was interviewed for the same software also indicated that it was a robust software and scored it as high. An additional scoring column was added to ensure that the correct total score was given for this specific case, as presented in Table 15. In this column, an additional score was given by the researcher based on the interview insights. Eventually, the total score now comes down to high, corresponding with the insights from the interviews.

3.4.2 Usability

Saher et al. (2020) defined specific quality characteristics for sustainable software systems for change management, of which usability is one aspect. The three given concepts of learnability, understandability and operability will be used for this thesis. Learnability was focused on learning to use the system and the efficiency of this, and understandability was focused on whether the software was suitable for the business needs. Operability relates to whether the system is easy to use and control.

In this thesis, the usability of various software tools will be reviewed through a system implementer's perspective on the implementation and use of ESG reporting solutions. Interviewees were asked to explain their experience with using and implementing the software in a client's organisation. In addition, implementers were asked to rank the statements presented in Table 11. Usability questions have only been asked to system implementers.

Table 11 Scale of usability

Concept	Indicator	Range				
Learnability	Learning to operate the systems is easy.	1	2	3	4	5

Concept	Indicator	Range				
		strongly disagree		neutral		strongly agree
Learnability	Learning to operate the system went quickly	1	2	3	4	5
		strongly disagree		neutral		strongly agree
Understandability	The system is unnecessarily complex	1	2	3	4	5
		strongly disagree		neutral		strongly agree
Understandability	The system is easy to navigate through	1	2	3	4	5
		strongly disagree		neutral		strongly agree
Operability	The system is easy to access.	1	2	3	4	5
		strongly disagree		neutral		strongly agree
Operability	The system is easy to use	1	2	3	4	5
		strongly disagree		neutral		strongly agree
Operability	The system is easy to implement in a client's organisation	1	2	3	4	5
		strongly disagree		neutral		strongly agree

In addition to the interviews, the usability scores from an external research platform have been used to extend the usability scores of this thesis. Usability scores for this platform were assessed through 2.5-hour live demonstrations and various questionnaires with vendors, users and industry experts. The concept of usability used by this research platform focused on the system's user-friendliness and the implementation services provided. The external research platform used a three-point scale to score software on their usability scores.

The total score for usability is derived a bit differently as it includes both the external research platform and interviewed implementers. An example of how these scores are combined is presented in Table 12. The total score from the external research platform has been recalculated to fit the five-point scale of this thesis. Eventually, both scores from the implementer and external platform are combined to form a total score for usability.

Table 12 Example of scoring usability

Sources	Usability x	y	Z	*External average (3-point scale)*	*Average (5-point scale)*	*Score*	*Total*
Implementer Software x	3	4	3		*3,3*	*Medium*	*Medium*
External research platform				*1,6*	*2,7*	*Medium*	

3.4.3 Matrix table

A matrix table with the results for flexibility, robustness, future proof and usability ratings is presented in Table 13. The software solutions are ordered based on their future proof score. Not all interviews were performed with both a vendor and an implementer. Therefore, for only three software both a vendor and an implementer were interviewed. Two of the three software tools that included both a vendor and implementer scored the same for flexibility and robustness. When interviews were conducted with two vendors or two implementers, contradicting scores were observed twice regarding robustness and three times concerning usability. In addition, one software did not receive a score for usability as no implementer was found, and the usability score was not available on the external research platform.

Table 13 Matrix table of the results for flexibility, robustness, future proof and usability ratings for each data source

Name	Flexibility					Robustness						Future proof	Usability			
	V1	V2	I1	I2	T	V1	V2	I1	I2	R	T	T	I1	I2	ER	T
Software 1	H		H		High	H		H			High	High	H			High
Software 2	H		H	H	High	H			H		High	High	H	H	M	High
Software 3			H	H	High			H	M	H	High	High	H	M	M	Med
Software 4	H				High	H					High	High			M	Med
Software 5	H				High	H					High	High			M	Med
Software 6			H	H	High			M	M		Med	Med	H	M	M	Med
Software 7	H	H	H		High	H	M	L			Med	Med	M			Med
Software 8	M				Med	H					High	Med			M	Med
Software 9	M				Med	H					High	Med				NA

Note: V1 is vendor interview one, V2 is vendor interview two, I1 is implementer interview one, I2 is implementer two, R is researcher, ER is external research platform and T is total. H is high, M is medium, L is low.

3.4.4 Software design features

Besides flexibility, robustness and usability, data collection also consisted of additional concepts related to the software tools' design, as presented in Table 14. A division has been made on whether these concepts have been included due to their relevance in theory or as advised by industry experts. Some concepts have been added after the interviews as part of an iterative research approach. The concepts that were included later on were mentioned during interviews as important enablers of either flexibility or robustness.

Table 14 Overview of additional concepts studied

Concept	Explanation	Included in accordance with
Heritage	With what type of business information systems, the vendors originally started with.	Theory
Architecture	Whether the software tool used a serverless, microservice, service-oriented, or monolithic architecture.	Theory

Concept	Explanation	Included in accordance with
Quality sub-characteristics	Which of the quality characteristics the software tool corresponded with.	Theory
Deployment model	Whether the software tool was offered on the cloud or on-premises, or a combination.	Theory
Cloud service methods	Whether the software tool was SaaS, PaaS, IaaS or a combination.	Theory
Offering	Whether the software tool was offered completely off-the-shelf, modifiable off-the-shelf, or a combination.	Industry Experts
ESG type	What type of ESG software the vendors offered, either focused on ESG reporting or a different type.	Industry Experts
Pricing model	Which type of pricing model the vendors used, ranging from subscription base, consumption base, or pay-as-you-go.	Industry Experts
Tenancy	Whether the software tool deployed a multi-tenant or single-tenant cloud, or a combination.	Interviews
Coding methods	Whether the software tool included low code/ no code	Interviews
Founded	The year in which the software tool was founded.	Interviews
ESG solution added	The year in which the ESG solutions has been added to the software tool.	Interviews

The definitions of various concepts mentioned in Table 14 have been provided in Table 15. The concepts have been identified both during the interviews and desk research. Vendors were asked during the interviews about the concepts that were derived from theory or that had been included due to the advice of industry experts. During the desk research, almost all concepts were identified as most vendors exactly described the concepts online.

Table 15 Definition of included concepts

Concept		Definition	Source
Quality sub-characteristics	Analysability	The ability of the software to detect the effect of a change or to detect reasons for failure.	(Saher, et al., 2020)
	Installability	The ability of a software to be installed in any type of environment.	
	Modularity	Components that do not have an impact on other components.	
	Replaceability	The ability of a system to be replaced by another product in the same environment.	
	Reusability	The ability of a software component or module to be reused in other systems.	

Concept		Definition	Source
	Interoperability	The ability of the software to be coupled to other software systems	
Cloud service methods	SaaS	Software as a service (SaaS) is an on-demand software type, such as an application that is provided on a subscription basis to users.	(Kavis, 2014; Odun-Ayo, et al., 2018)
	PaaS	Platform as a service (PaaS) is a cloud service where users can host their own applications on.	
	IaaS	Infrastructure as a service (IaaS) provides virtualised resources such as servers, storage and networking infrastructure.	
Offering	Off-the-shelf	The software is used "as-is"	(Carney & Leng, 2000)
	Modifiable off-the-shelf	The software can be modified or customised after it has been purchased	
	Custom development	The software is specifically developed for the specific business need	
Pricing model	Subscription base	The number of usage and services are fixed beforehand	(Bhargava & Gangwar, 2016; Wu, et al., 2019)
	Pay-as-you-go	Billed based on usage afterwards	
	Consumption base	Credits are bought in advance and users pays for the amount of usage	
Tenancy	Multi-tenant	Multiple users sharing the same database	(Mietzner, et al., 2009)
	Single tenant	Each customer has their own instance and a separate database	

3.5 Data analysis

Throughout the study, several qualitative data sources were used for the data analysis. The data were analysed through a matrix table to generate one dataset and create an overview of the data to allow detailed analysis of the selected cases (Miles, et al., 2014). The data analysis of the desk research and interviews will be executed differently.

3.5.1 Desk research

During this phase, the collected data from the desk research was structured and categorised. During the desk research phase, concepts such as the architecture, the quality sub-characteristics, cloud service methods, deployment model, pricing model, tenancy, coding methods, and offering were sought. These concepts were often literally mentioned on the vendor's website, on community platforms or in webinars. In most cases, vendors published the included concepts online, but sometimes this was not the case. Hence, interviews were employed as a means to verify and validate the gathered data. The findings were systematically assigned to each software vendor and eventually incorporated into the matrix table. An overview of the used data sources per software case is presented in Table 13.

3.5.2 Interviews

During the interviews, interviewees were asked various Likert scale questions and were asked to provide more in-depth explanations behind the given scores.

3.5.2.1 Semi-structured part

Interviewees were asked to explain and justify the assigned flexibility, robustness and usability scores. The interviews were recorded and transcribed to ensure that all data was captured. The qualitative data analysis software NVivo 20 was used to code the transcripts and structure the findings. A Thematic inductive coding approach was utilised. This type of coding process starts with identifying and labelling text sections into codes, and similar text sections are added to the same codes (Bryman, 2012). Where possible, the categories were connected to the Likert scale questions to make it easier to complement these results with in-depth explanations. All interviews were recoded for a second time to ensure that no codes were missing and to include new concepts from the iterative process. Eventually, the codes were merged and linked when codes touched the same subject to reduce overlap. A complete overview of the codes can be found in Appendix C.

Additional findings emerged from the interviews. Vendors and implementers discussed other possible influential factors related to the software solutions' flexibility and robustness. These factors were added to a table with an overview of all results, which can be found in Appendix D.

3.5.2.2 Structured part

The Likert scale questions were asked to identify possible relations between architecture and the future proof level and usability and flexibility scores. The scores from each respondent have been averaged and merged with other sources to create a total score for flexibility, robustness, future proof and usability for each software solution. The insights from the in-depth interview questions and Likert scale scores have been combined in a total results table in Appendix D.

A matrix table containing the scores from each data source in combination with the characteristics was used. The first phase of the interpretation is related to the presentation of the architectural styles and their corresponding quality characteristics. Then the first proposition stated in the theoretical framework was compared based on the flexibility and robustness of each software in combination with the architecture type. Additional factors that might relate to the future proof scores of the software were examined, and similarities in the data were interpreted. These factors were mentioned by interviewees in the interviews and, later on, added to rule out potential alternative explanations for future proof scores. Following that, the results for usability and the accompanying proposition were discussed on eventual patterns. Eventually, the propositions and reasons behind the differences are presented in the discussion.

3.6 Research quality indicators

3.6.1 Reliability and validity

By employing predeveloped software characteristics that have been tested and used by other studies, construct validity can be achieved. However, some other issues relating to sampling, data collection and analysis can occur. The sampling's techniques validity is limited as non-probability sampling is used, and the vendors have been contacted from a pre-selected list. Due to this, a possible sampling bias can occur as these vendors have been recognised as the most renowned software vendors, overlooking smaller vendors, which could influence the eventual future proof scores. Another issue is that not all data can be collected through observations or desk research, and therefore, interviews with software vendors were included. This increases the likelihood of manipulation as these vendors might overestimate their own software. To reduce this, system implementers were interviewed, and external market research was used to combat this. In addition to that, the scores and in-depth explanations were compared to ensure that these were consistent. This is also known as data triangulation which increases the validity of the research (Bryman, 2012). One limitation of this triangulation is the eventual bias if certain software tools include an extra verification step and others do not. Therefore, in cases where no vendor could be reached, two implementers were interviewed, of which at least one needed to have a software architecture background.

4 RESULTS

This chapter will present the results of the collected and analysed data. First, background information on the collected software tools will be provided to introduce the various cases. The scores for flexibility, robustness and usability are also given for each case, including a short explanation for each of the scores. The second part of the results will focus on future proof scores, patterns, and data similarities between the architecture, related quality sub-characteristics, and flexibility and robustness. In order to rule out potential alternative explanations and factors that may relate to the future proof scores, additional software design features were examined. These included factors such as the type of cloud service methods, coding methods, offering, and the type of ESG software, which will be further discussed in subsequent sections. Eventually, patterns between usability, flexibility and future proof are discussed to understand their relations. An overview of all results can be found in the table in Appendix D.

4.1 Profile description software cases

A total of nine ESG software solutions were assessed in this study, of which four out of the nine software tools originally started as GRC software, three as ESG software and the remaining two as BPM or EHS software. The majority of the ESG software solutions focus on reporting, whereas two software solutions differentiate from this and focus mainly on carbon management and LCA. Most vendors have launched their ESG solution in the past four years, whereas other vendors have been in this industry since the early 2000s.

Software 1

Software 1 originally started as a GRC system but has expanded over the years to fairly all types of business information systems, including an ESG reporting solution. The software is priced according to subscription bases and is offered to users in three ways: off-the-shelf, modifiable off-the-shelf, and it can be custom developed. A combination of service-oriented and microservice architectures are used with on-premises, single-tenant cloud and multi-tenant cloud deployment models. The software tool scored high for flexibility. The main reasons for this high score are related to the fact that the solution is built on a platform where users can adjust the solution without coding. Even though several regulations are offered out of the box, users can also add customised reporting formats. Robustness scores are also high as issues are aimed to be resolved instantly, and the service teams are located in every continent of the world to ensure that someone is always working on the issue. Overall, the ability to be future proof for this software is high; this is also reflected by other interviewees who mentioned Software 1 as an example of flexible and robust software. In terms of usability, the software also received a high score. The usability scores are high as users are already familiar with the interface and software, as most users will use the ESG solution as an add-on from the original GRC system.

Software 2

Software 2 started as ESG reporting software. The software is priced according to a consumption-based model and is offered to users as an off-the-shelf software. A combination of a microservice and serverless architecture is used for the tool in combination with a cloud deployment model. Flexibility scores are high as regulations and reporting features can be adjusted instantly, but the customisation of functionalities needs to be done by the Software 2 team. Robustness is also scored as high, but it must be noted that the system implementers have tested the tool and worked with it by themselves but did not implement it in a client's organisation. Therefore, the scores for bugs and outages were often marked as not applicable. Overall, the ability to be future proof for this software is high. The usability score was also marked as high by the two implementers and medium by the external research platform.

Software 3

Software 3 was originally an EHS software that added an ESG reporting solution to its software. The software is priced according to a subscription-based model and is offered to users as either an off-the-shelf or modifiable off-the-shelf software. A combination of a microservice and serverless architecture is used for the tool in combination with a cloud deployment model. The software scored high for flexibility as it can be configured, and most factors can be adjusted by the system implementers. According to the two implementers, the software is marked as one of the more robust software tools, resulting in a high robustness score. Therefore, the ability to be future proof for this software is high. Usability was scored as a medium, and the main reasons given for this related to the intuitiveness and complexity of the software. However, there was a difference between the two implementers as one scored usability as high and the other as medium. The score from the analysis platform was also included for this software and came down to medium.

Software 4

Software 4 is a BPM system that initially focused on EHS, asset management and quality management, and recently they added an ESG reporting solution. The software is priced as a pay-as-you-go depending on the usage and is offered as modifiable off-the-shelf. The software uses a service-oriented architecture with on-premises or multi-tenant cloud deployment models. The software scored high for flexibility, which is mainly related to the no-code platform, where users can adjust the platform without needing to code. As most changes are possible to adjust in the configuration layer, the process of adjusting the software is rapid. Robustness scores are high as well. As the ESG solution has just been added recently, robustness scores were mainly based on the overall platform, as there were not enough use cases available. Most bugs that occurred related to the lack of testing on the user's end but not necessarily to the software tool's abilities. Overall, the software solution's ability to be future proof is high. The score for usability was also derived from an analysis platform and was scored as a medium based on a combination of data on the user-friendliness of the software, the number of languages that were provided and the software support. Software 4 can mainly improve its training offering, as they do not offer out-of-the-box training for users.

Software 5

Software 5 started as a GRC system and later added an EHS and ESG reporting solution to the software. The software is priced according to a subscription base and is offered to users as a modifiable off-the-shelf model. The software uses a service-oriented architecture with a single-tenant cloud deployment model. Flexibility scores are high as users can change a large part of the out-of-the-box solution by themselves without interference from the software vendor. Therefore, changes can be made relatively fast. Bugs relating to data are taken very seriously, and the robustness of the software is, therefore, scored as high. Overall, the ability to be future proof for this software is high. The score for usability was also derived from an analysis platform and was scored as medium. The software is offered in a large variety of languages to support various types of end users.

Software 6

Software 6 originally started as a GRC software and, later on, added an ESG reporting solution. The software is priced according to subscription and consumption bases and is offered to users in three ways: off-the-shelf, modifiable off-the-shelf and custom development. The software uses a service-oriented architecture with a cloud deployment model. Flexibility scores are high as the software is highly customisable, and almost anything is possible to be adjusted. However, due to this high customisation, the robustness of the software is lacking and is impacting the performance of the software, resulting in a medium score for robustness. *The implementer of this software advised that Software 6 should set* *boundaries for the amount of customisation that can be made to ensure that the performance is not affected.* Overall, the ability to be future proof for this software is medium. Usability scores are medium as the software is not as intuitive as other software, but there is the possibility to customise it completely to make it more intuitive for end-users.

Software 7

Software 7 started as a GRC system and added an ESG reporting solution to their current GRC software. The software is priced according to subscription bases and is offered to users in three ways: off-the-shelf, modifiable off-the-shelf and can be custom developed. A combination of all architectures is used for the platform with the deployment models on-premises and single-tenant cloud deployment. The software scored high for flexibility as changes are implemented fast, and most changes can be accomplished by small configurations. Robustness is scored as medium, as most changes that were implemented resulted in bugs, and clients have had considerable downtime. Besides that, most bugs that occur for an older release are reoccurring for newer releases as newer releases are not tested properly for all bugs. The service team is only located in one continent, which relates to issues with communication and the amount of time that is worked on solving bugs. So, where the software is able to quickly adjust new functionalities, the quality that it delivers is lacking. Overall, the ability to be future proof for this software is medium. The usability of the tool is "not worse than other GRC tools" (Implementer Software 7) and is straightforward for users that are a bit familiar with it; therefore, it is scored as medium. The main issue for usability is related to implementation in a client's organisation, as support and knowledge from Software 7's side were missing.

Software 8

Software 8 is an ESG software with a main focus on carbon accounting. The software is priced according to subscription bases and is offered to users as a completely off-the-shelf product. The software uses a serverless architecture with a multi-tenant cloud deployment model. Flexibility scores are medium as users cannot modify any aspect by themselves and must wait for the software vendor to adjust changes. Due to this, users will need to wait on new releases to have certain functionalities incorporated. However, not all requested functionalities will be added as it needs to bring a certain value to the software. Robustness, on the other hand, is high as the architecture is built to resolve bugs quickly. Overall, the ability to be future proof for this software is medium. The score for usability has also been derived from an external research platform and was scored as medium. The score was based on the implementation and training services that were offered and the user-friendliness of the software.

Software 9

Software 9 originally started as an ESG system focusing mainly on Life Cycle Assessments and compliance. The software is priced on a subscription basis and is mainly offered to users as an off-the-shelf model. However, it is possible to request a modification of certain aspects, but this is less common for this software. The software uses a microservice architecture with deployment models on-premises and cloud deployment. Flexibility is scored as medium as changes in the software are mostly adjusted for all users. Thus, users cannot easily adjust aspects of the software by themselves. Changes will be implemented with new releases, which do happen biweekly. Robustness, on the other hand, is scored high as testing is an important part of the software resulting in the goal of zero downtime and a low number of bugs. Overall, the ability to be future proof for this software is medium. No usability scores are present for this software.

4.2 Future proof

Based on a combination of the semi-structured and structured questions, the determination of the extent to which an ESG software solution was future proof was examined. As discussed in the methods section, a mean score for future proof was computed by averaging interviewee responses on the flexibility and robustness of the software tools. These scores were supported by the explanations given during the interviews. The results and the scores from each participant are presented in Table 13, and the breakdown of future proof, flexibility and robustness scores are presented in Figure 6.

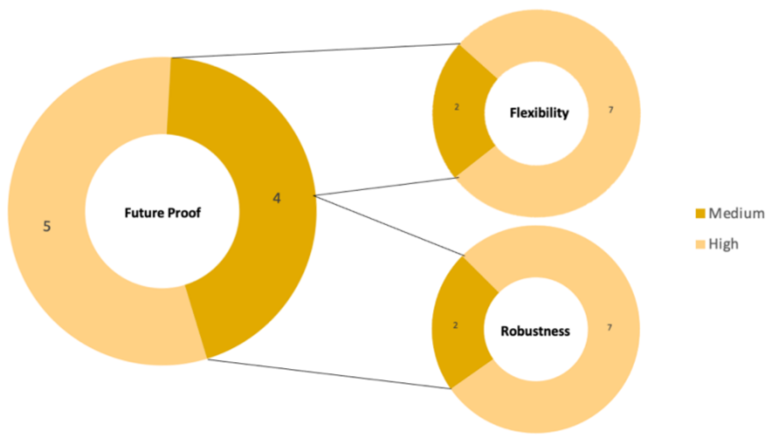


Figure 6 Division of future proof, flexibility and robustness scores

There were several notable observations identified. First, the scores for the concept of future proof are rather high. Out of nine software tools, five scored high for future proof, and four scored medium. No software tool received a low score for future proof. Second, when looking more closely into flexibility and robustness, two software tools scored medium for flexibility and two for robustness. None of the software tools received a low total score for either flexibility or robustness, and none of the software received both a medium flexibility and robustness score. Third, implementers not only gave medium scores, but some vendors also scored their tools as medium. Indicating that vendors would not always score their software as high.

The importance of flexibility and robustness of the software tools has been mentioned by interviewees as well. In terms of flexibility, the need for adjustable software was recognised by multiple interviewees and was also explained well by the vendor from Software 4 "*The ESG landscape, especially with like scope 3 emissions, these are changing all over the place and so the requirements of what the tool needs to do to accommodate are also always evolving*.". In addition, adjusting a regulation format was not necessarily a problem as multiple software did not use out-of-the-box frameworks but allowed users to customise the necessary data fields and workflows. These custom frameworks were mentioned to be used due to the evolving landscape and to be able to respond quickly to new regulations. Especially since “*time is not a luxury that customers have*” (Software 2 Implementer 2)

In terms of robustness, multiple interviewees mentioned that “*bugs are a natural occurrence*” (Vendor Software 5), and are part of software development. Testing is an important aspect of avoiding bugs, and the importance of avoiding bugs varies across vendors. In terms of ESG and new regulation frameworks, robustness problems would not necessarily happen for these types of changes. However, ESG is very data-driven, so bugs would mainly relate to data collection of the software, as explained in the following quote by the vendor from software 4 “*If the data would change, you have to get something new in from a system that we have not done before. There might be a bug in terms of that it doesn’t load properly*”.

4.3 Software design features

Each software tool had a different composition of design features. An overview of some of these features is presented in Table 16. It will be discussed later on how each of these software design features relates to future proof, flexibility and robustness. First, the architecture styles will be explained, followed by additional features and insights.

Table 16 Overview of software tool features

Name	Heritage	ESG Type	Offering	Deployment model	Architecture	Tenancy	Cloud service methods	Coding methods	Founded	ESG solution added
Software 1	GRC	ESG Reporting Software	Modifiable off-the-shelf & Custom developed	Cloud, On-premises	SOA, MSA	Multi- & Single-tenant	PaaS, SaaS	Low code, no code	2004	2021
Software 2	ESG	ESG Reporting Software	Off-the-shelf	Cloud	MSA, Serverless	Multi-tenant	SaaS		2004	2004
Software 3	EHS	ESG Reporting Software	Modifiable off-the-shelf	Cloud	MSA, Serverless	Multi-tenant	SaaS		2000	2017
Software 4	BPM	ESG Reporting Software	Modifiable off-the-shelf	Cloud, On-premises	SOA	Multi-tenant	SaaS	Low code, no code	2005	2022
Software 5	GRC	ESG Reporting Software	Modifiable off-the-shelf	Cloud, On-premises	SOA	Single-tenant	PaaS	Low code, no code	2008	2021
Software 6	GRC	ESG Reporting Software	Modifiable off-the-shelf	Cloud, On-premises	All	Single-tenant	SaaS	Low code, no code	1999	2021
Software 7	GRC	ESG Reporting Software	Modifiable off-the-shelf & Custom developed	Cloud, On-premises	SOA	Multi- & Single-tenant	SaaS		2004	2022
Software 8	ESG	Carbon Management Software	Off-the-shelf	Cloud	MSA, Serverless	Multi-tenant	SaaS		2020	2020
Software 9	ESG	Product Lifecycle Assessment Software	Off-the-shelf	Cloud	MSA	Multi- & Single-tenant	*Becoming SaaS*		2001	2001

Note: MSA = Microservice architecture, SOA = Service-oriented Architecture

4.3.1 Architecture

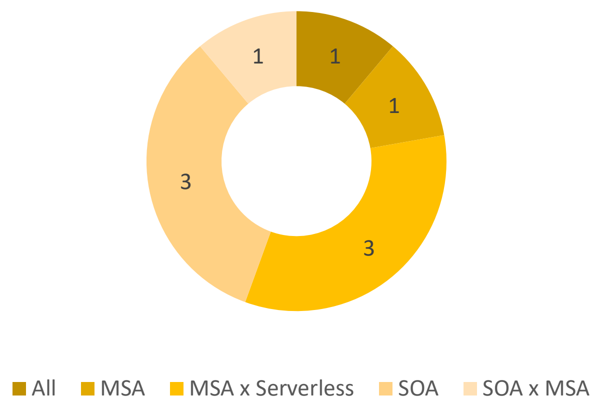


Figure 7 Number of software cases per architecture style

During the desk research and interviews, five distinct architecture variations were identified, with three of them being combinations of different architectures, while the remaining two were standalone architectures, as presented in Figure 7. Among the nine software solutions analysed, three utilised the service-oriented architecture, while another three vendors employed a combination of microservice and serverless architectures. The remaining three software solutions were built on a microservice architecture, a combination of service-oriented and microservice architectures, and a combination of all architectures, including the monolithic approach. Interesting to note that all software at least included a service-oriented architecture or microservice architecture.

The inclusion of the quality sub-characteristics analysability, installability, modularity, replaceability, reusability and interoperability were examined through desk research and in the interviews. These quality characteristics were deemed to be important for ESG software. It was furthermore expected that software with a service-oriented or microservice cloud architecture included all these quality characteristics. During the interviews, some vendors already mentioned that the software tool contained certain quality characteristics without being explicitly asked about it. In other instances, the characteristics were explicitly asked. Almost all software contained the requested quality characteristics. For one software solution, not enough information was available online, and the implementer was not sure about certain characteristics. The eight other software tools likely contained all quality characteristics as they all include either a service-oriented or microservice architecture in combination with cloud. According to the cited literature, these quality characteristics should be present for these types of architectural styles, which corresponds with the results found. In addition, the vendor from Software 7 mentioned how these quality characteristics are fundamental aspects of the software tools. An overview of the quality characteristics of each vendor is presented in Table 17. No strong pattern emerges as there is no variation in the outcomes for these characteristics but a variation in future proof scores.

4.3.1.1 Future proof and architecture

According to previous literature, serverless, microservice and service-oriented architectures would likely relate to more flexible and robust software, whereas a monolithic architecture was considered to not relate to high flexibility and robustness. Consequently, the first proposition was formulated as follows:

Serverless, Microservice, and Service-oriented architecture styles will likely relate to future proof software systems, whereas Monolithic architecture will likely not relate to future proof software systems.

To determine whether proposition one is supported, an examination of the future-proof scores and the corresponding architecture styles is provided, as presented in Figure 8 and Table 17.

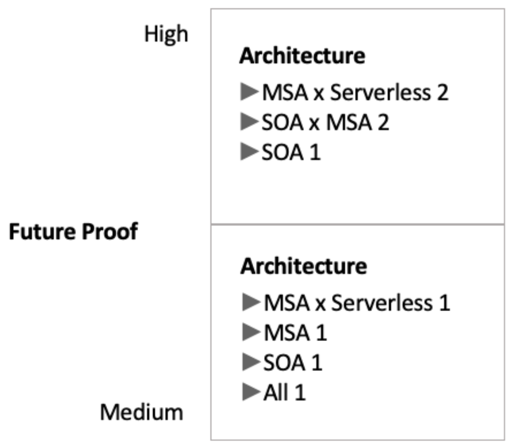


Figure 8 Future proof score and architecture styles

Five software tools support the proposition, as they include a service-oriented, microservice or serverless architecture and score high in terms of future proof. Nevertheless, the software tool that scored medium for future proof also included these architecture styles. There was only one tool software with a monolithic architecture, but it was combined with all other architecture types as well. So, it is unclear to what degree monolithic architecture contributed to this medium score, as the software was a combination of architectures and not solely monolithic. Thus, it seems like there is no clear pattern between the architecture styles and future proof scores, as serverless, microservice and service-oriented software both received high and medium scores for future proof.

Table 17 Matrix table of the results for architecture, deployment model and quality characteristics

Name	Architecture	Deployment model	Quality characteristics	Flexibility	Robustness	Future proof
Software 1	SOA, MSA	Cloud, On-premises	All	High	High	High
Software 2	MSA, Serverless	Cloud	All	High	High	High
Software 3	MSA, Serverless	Cloud	All	High	High	High
Software 4	SOA	Cloud, On-premises	All	High	High	High
Software 5	SOA	Cloud, On-premises	All	High	High	High
Software 6	SOA	Cloud, On-premises	NA	High	Medium	Medium
Software 7	All	Cloud, On-premises	All	High	Medium	Medium
Software 8	MSA, Serverless	Cloud	All	Medium	High	Medium
Software 9	MSA	Cloud	All	Medium	High	Medium

Note: MSA = Microservice architecture, SOA = Service-oriented Architecture

When examining the concepts of flexibility and robustness individually, there is hardly a noticeable pattern, as presented in Figure 9. The two software that scored medium for flexibility had either a microservice architecture or a combination of microservice and serverless architectures. These architectures were also utilised by software that scored high for flexibility. Regarding robustness, a slight pattern emerges, as five out of six software with a microservice architecture, either in combination with another architecture or on its own, resulted in high robustness. The two software tools with serverless architecture also scored high for robustness, except for the software with all architecture styles.

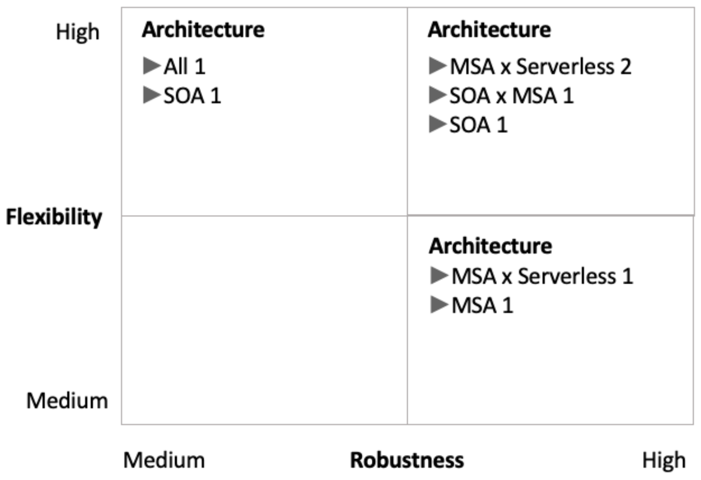


Figure 9 **Overview of the relationship between flexibility, robustness and the architecture styles**

Hence, based on the results of this thesis, it can be concluded that proposition one is not supported. The adoption of a microservice or service-oriented architecture style did not necessarily relate to a future-proof ESG software system. The architecture style alone cannot be considered the sole determinant of a software system's flexibility or robustness, as indicated by the findings from the interviews. During the interviews, participants did not explicitly mention these architecture types as primary drivers of flexibility and robustness. Instead, other factors such as the utilisation of cloud service methods, coding methods, software offerings, and the type of ESG software were deemed more important in influencing flexibility and robustness.

4.3.2 Additional features

Within this section, additional influences that potentially relate to the concepts of this study are considered, providing a comprehensive understanding of the research findings. These findings emerged both through consultation with industry experts and during interviews with vendors and implementers. A lot of different factors were indicated. For heritage, deployment model and tenancy, no relation emerged with flexibility, robustness or future proof. The cloud service methods, coding methods, offering and ESG type, did seem to demonstrate a possible relation between flexibility, robustness or future proof in general. First, insights that did seem to result in possible patterns will be discussed. Eventually, the insights that did not seem to have a relation to flexibility, robustness or the level of future proof are briefly discussed, as these factors were still deemed important by the interviewees.

4.3.2.1 Cloud service methods

The cloud service methods, software as a service (SaaS) and platform as a service (PaaS) were mainly used, as presented in Figure 9. SaaS is a cloud software used as an application, whereas PaaS is a platform where applications can be hosted and managed on. One software was still in the process of becoming SaaS and has therefore not been considered in Figure 10.

An interesting finding emerged from software tools with a PaaS cloud service, as the two software that used this service both scored high for future proof. The benefits of PaaS have also been acknowledged by the vendors that are offering these services. The vendor from Software 1 argues*, "that is where the prominence of platform is very key and important, because these features could change there. Could be new set of applications coming out in the future, but with the platform it is capable of spinning this off very quickly.".* This interviewee thus indicates that a PaaS platform is able to release new applications quickly. In addition to that, the implementer from Software 1 also suggests that the use of PaaS makes the process of ESG data collection more easily, as explained by the following quote: "*It is leveraging the* *same platform as all the other modules. Which means that if I want to collect data on ESG, I have data from everything else already in the platform.".* The absence of a PaaS service does not necessarily mean that a software tool cannot score high for future proof, as three out of six software that used a SaaS service also scored high for future proof.

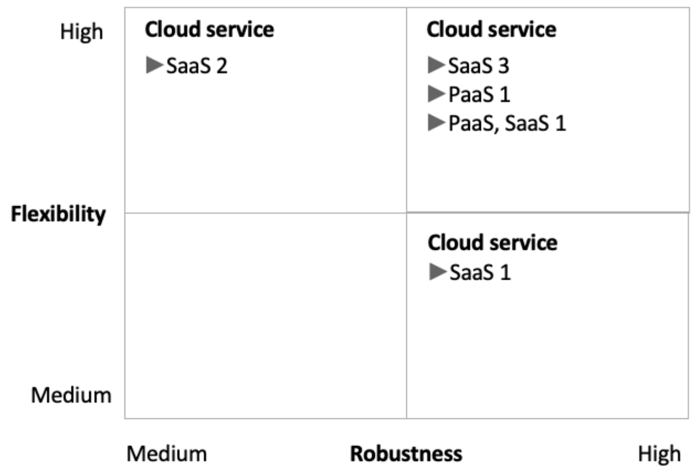


Figure 10 Overview of the relationship between flexibility, robustness and cloud service methods

The results suggest that software tools that offer a PaaS cloud service are possibly related to higher scores for both flexibility and robustness. It must be mentioned that there were only two cases in this study that contained this cloud service method.

4.3.2.2 Coding methods

In addition to the cloud service, the coding methods show a possible relation to flexibility scores. The presence of a low code, no code method emerged as a possible factor for higher flexibility. This specific coding method enables adjustments without the requirement of programming skills. This method reduces development time and minimises the need for user programming input. Consequently, end-users find it easier to make modifications to the software. All software instances that incorporated a low code, no code method scored high in terms of flexibility, with three out of four also achieving high future proof scores. However, it should be noted that the absence of this method did not necessarily result in lower scores.

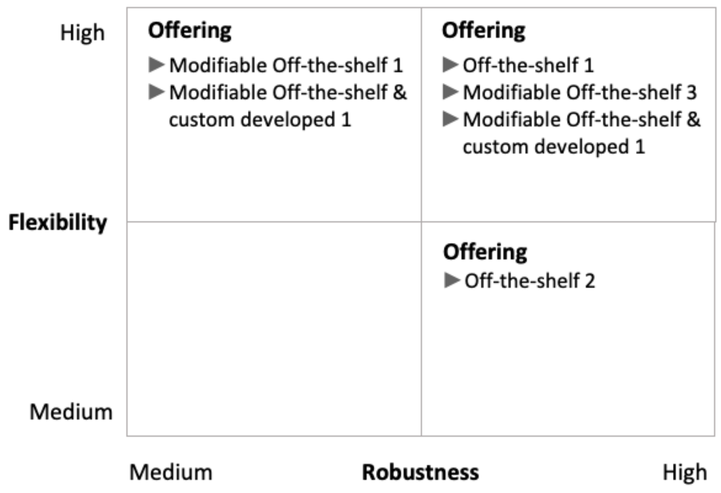
The following quotes from various vendors also explain the advantages of this type of method. The vendor from Software 4 proposed, *"We make the configuration changes, which is no code, so we can very rapidly, you know, turn that around ..." "So we're able to meet with these large customers, understand their process and really very rapidly configure a solution for their needs without touching code, without writing Java and. The platform will handle all of the business logic.".* The vendor from Software 5 added, *"… that's the fastest route. If you can use our configuration tools or our design toolkit, to be more precise, they can do these changes in less than five days, depending on the size of the changes. But that's one of the reasons why our customers love our software, especially large enterprise customers because they are in control of a lot of the things, they don't have to wait on us. If they wanted to add new fields, someday tomorrow, you know there's a new regulation and suddenly they have to have this new workflow. They can do that themselves, …".*

The results suggest that software tools possibly score higher in terms of flexibility and future proof by utilising a low code, no code method. Software tools offering a low code, no code method are able to respond faster to changes and allow the end-user to make certain changes to the software by themselves. It is worth noting that the absence of this method did not necessarily result in lower scores for flexibility and future proof.

4.3.2.3 Offering

The software instances were offered in various ways, either completely off the shelf, modifiable off the shelf or with the possibility for custom development. Software that is modifiable off-the-shelf is essentially an off-the-shelf product, but it can be modified.

Several observations can be made from the findings presented in Figure 11, which will be further explained by quotes from the interviews. First, five out of nine software that scored high for flexibility offered either modifiable off-the-shelf or custom-developed software. Second, off-the-shelf software tools scored twice medium for flexibility and once high. Medium robustness scores were only perceived for either modifiable or custom-developed software, but modifiable or custom-developed software scored four times high for robustness. The findings will be further substantiated with interview quotes.



High
Offering
▶ Modifiable Off-the-shelf 1
▶ Modifiable Off-the-shelf & custom developed 1
Offering
▶ Off-the-shelf 1
▶ Modifiable Off-the-shelf 3
▶ Modifiable Off-the-shelf & custom developed 1
Flexibility
Offering
▶ Off-the-shelf 2
Medium
Medium
Robustness
High

Figure 11 Overview of the relationship between flexibility, robustness and system offering

Software that is modifiable off the shelf can be adjusted after an end-user buys the software, whereas custom development is completely built from scratch and designed specifically for the end user. The higher flexibility scores for these two offering types can be explained by the fact that these software tools are easier to configure and either already offer out-of-the-box solutions that can be adjusted, or they could completely design a functionality for the client. This is also explained by the vendor from Software 4, *"So we have what we call just a starting point or off the shelf, and a lot of companies will insist that that's all they want. But we always include some professional services to set up the product. Then through some engagement, there's very often, more often than not, quite a bit of configuration in that no code environment to fine-tune exactly what they'd like to see*.". Suggesting that clients often want to adjust and configure a software tool after they already bought the off-the-shelf variant. Therefore, certain vendors allow the off-the-shelf variant to be modified. The implementer from Software 6 also indicated that *"90% of their clients are happy with the capabilities provided by the software. So, it does cover a lot of things."*. However, *"… 'Software 6' provides the ability to actually enhance their solution. It's not like the vendors who need to go and implement this from scratch."*.

Furthermore, two out of three software with an off-the-shelf offering scored medium for flexibility, as presented in Figure 10. This could be explained by the fact that off-the-shelf software only offers solutions that are accessible to all users. The vendor from Software 8 also explains this with the following quote "*Right now, it is completely off the shelf. So, all of our users use the same product more or less*.". Even though flexibility might be lower for off-the-shelf, Software 9 said to have the following reason for that "*We try to stay as a product as much as we can. Because we want to make it so that each of our customers can profit from our further developments, we want to be a product company. And for that, it's important to do not too much customising*.". This also applies to Software 8 as they explain how they will adjust all functionalities for all users "*And what we're finding right now is that most of the time when one of them requests functionality it applies to everyone...*"

Software that is offered completely off-the-shelf scored high for robustness in all three instances. Whereas software that is custom developed or modifiable off the shelf for two out of four software scored medium for robustness. It was highlighted in the interviews that custom-developed software tends to give more robustness problems. According to the implementer for Software 1, "*the only bugs we can see are linked to the custom development.*". Software 6 agrees that there are problems with custom development and explains it as follows "*So yes, we have encountered performance problems as a part of this implementation because it was a very custom code, and it is a very complex code*.".

The results suggest that software that is modifiable off the shelf or can be custom developed possibly relates to higher flexibility scores, whereas software that is completely off the shelf possibly relates to higher robustness scores. This does not necessarily mean that modifiable or custom-developed software tools are less robust, as four out of six software tools that offered this were perceived to be highly robust.

4.3.2.4 ESG type

In terms of the specific ESG type, there were three different types of ESG software included in this study, of which seven out of nine had a main orientation on ESG reporting, and two software had a different main focus. One of them was a carbon accounting software tool, and the other was an LCA and compliance software tool. In terms of flexibility, the two differentiating ESG software received a medium score, as presented in Figure 12. The difference in these scores can be explained by the extensibility of adjusting reporting and functionalities. Both of these software vendors change functionalities for all users and include detailed calculations to accompany new regulations.

There are also differences in how ESG reporting solutions are offered for each vendor. In some cases, an agnostic approach was offered, meaning that they do not literally define all metrics. The vendor from Software 5 explained that *"... if a customer is doing GRI, it's probably about 60% there for CSRD*". Therefore, these vendors do not include all questionnaires separately, as there is a lot of overlap between ESG regulations. Other software solutions did include out-of-the-box frameworks from separate regulations. A downside of this is in cases when vendors want a subscription for a specific ESG framework, as it can take more time to adjust. As explained by the vendor from Software 7, "*What takes long is actually connecting to, like, SASB, for example. Speaking to their team for a subscription that in itself takes more time. At least when we initially did SASB, that was my experience with trying to get them onboarded and get them to agree to a partnership and all that.*". Meaning that out-of-the-box frameworks in these cases could impact flexibility.

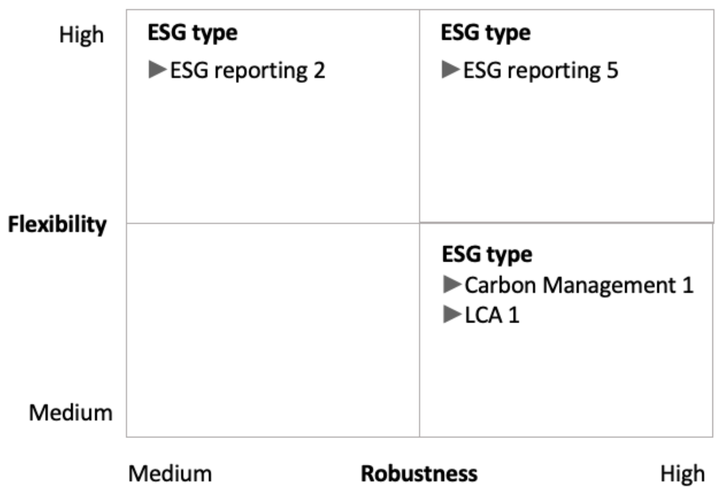


Figure 12 Overview of the relationship between flexibility, robustness and ESG type

Thus, there seems to be a possible relation between the specific ESG type and how detailed and specified the ESG calculations are and the flexibility of the offered software solutions.

4.3.2.5 Additional insights

Certain factors that were mentioned during the interviews did not necessarily seem to relate to the scores for flexibility, robustness or future proof. These factors relate to software heritage, deployment model and tenancy.

First, the heritage of the software. The software examined started from various origins before the ESG solutions were added. Some software tools were initially developed as ESG or EHS software, whereas others were regular business information systems, as presented in Figure 13. In the interviews, it was mentioned how software that was not originally created for ESG tends to be more flexible, as explained by the implementer from Software 2 "*So, 'Software 1', if I just want to reflect, then it allows a bit more flexibility that way because 'Software 1' wasn't created for ESG, right? So, it certainly gives you more flexibility sometimes"*. However, no clear pattern emerged in the results.

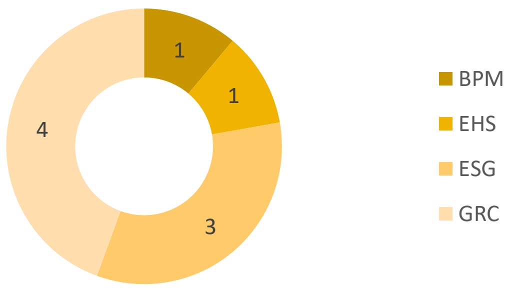


Figure 13 Count of software heritages

In addition, another possible factor that was mentioned during the interviews was the specific deployment model. Both on-premises and cloud deployment models were considered. On-premises refers to desktop applications, while a cloud model is web-based. Some interviewees related on-premises with possible robustness problems. The implementer of Software 7 shed light on a possible explanation for these lower robustness scores: "*I think they are faster for the cloud, so the problems are for the on-prem. I think they can fix the bugs faster on the cloud*.". The implementer from Software 3 also suggested that certain clients switch to a cloud model as these are generally faster than on-premises models. None of the participating software vendors employed solely an on-premises deployment model. Therefore, distinguishing the precise impact of on-premises on robustness is challenging, and no clear pattern emerges.

The cloud deployment model can be further classified as either multi-tenant or mono-tenant. Multi-tenancy implies multiple users sharing the same database, while mono-tenancy means that each customer has their own instance and a separate database and is also referred to as single-tenant. During the interviews, the interviewees predominantly emphasised how a mono-tenant configuration contributed to higher robustness. This is also illustrated by the following quotes from the vendor from Software 1 *"It's a mono-tenant organisation. Meaning that every customer has its own technical environment. So you don't have to put the system down because you need to, you know, do something on another tenant which has a problem or something like that, it never happens." "Each customer has its own environment, unique mono tenant, which means as well if you want to migrate to a new data centre, it's very easy as well because you take the tenant here, to mirror as well to have backups etc. It's very, very simple, so it's a bit more expensive of course. To do that like this, but in terms of security, reliability and resilience. "*This was also reflected by the implementer from Software 3, that worked with a multi-tenant software that scheduled weekly downtime to work on the platform. *"... there are scheduled downtimes, and some of them are during the business day, but they aren't as a result of changes. So just as a result of the normal working schedule...".* The interviewees thus suggest that single tenancy would relate to higher robustness and flexibility. However, this pattern does not necessarily emerges in the results.

4.3.2.6 Summary of results

The study's additional findings provide insights into various factors that potentially relate to flexibility, robustness, and future-proof ratings of the software solutions. The analysis considered both factors that showed a relation and additional factors without a relation that were mentioned in interviews. As presented in Table 18, only five possible patterns emerged. Most of the time, these patterns were linked to the design of the case and were observed for flexibility. In most cases, a factor led to a positive effect, but the absence of this factor did not necessarily lead to a negative effect. In addition, the chances of patterns in the data were generally smaller as variation in the outcomes for robustness and flexibility was limited; only two software scored medium for flexibility and two for robustness.

The most interesting patterns related to the specific cloud service methods, methods of coding, offering and ESG type. It seemed that software that used the specific cloud service PaaS related to high robustness and flexibility score for the two cases. But this did not exclude software with SaaS from scoring high for both robustness and flexibility. The same accounted for the presence of a low code, no code method, as this was related in all four cases to high flexibility. Software without low code, no code, also scored high for flexibility, but not consistently. The software offerings modifiable off-the-shelf and custom development options related to high flexibility scores for all cases. Furthermore, ESG software with a more specified focus demonstrated lower flexibility than general ESG reporting software. These findings highlight that there are possibly more factors than just architecture that could relate to flexibility, robustness and the future proof level.

Table 18 Table of possible patterns between flexibility, robustness, future proof and additional factors

	Flexibility	Robustness	Future Proof
Offering	X	X	
Heritage	/		
ESG type	X		
Pricing model			
Deployment model		/	
Architecture			
Quality characteristics			
Tenancy		/	
Cloud service methods			X
Coding methods	X		
Founded			
ESG solution added			

Note: An X indicates that there is a possible pattern, a / indicates that an interviewee mentioned this as a possible explanation for flexibility or robustness

4.3.2 Usability

Eventually, the usability of the software was examined. As stated in the theoretical framework, usability is an important aspect of a software tool as it will influence the use and experience of users by learning and using the software. Relating to the usability scores for the nine ESG software cases. None of the software scored low for usability, but only two out of eight software scored high for usability. The other software tools scored medium for usability. For one software, no usability score is present as no implementer was found, and the usability scores were not available on the external research platform.

Previous literature mentioned that flexibility and usability are linked, as inflexible software would likely lead to low usability of the software as the system is not flexible enough to adjust to a user's wishes. Thus, the second proposition was formulated as follows:

A low flexibility rating and, thus, a low future-proof rating will likely relate to low usability.

To determine whether proposition two is supported, an overview of the future proof, flexibility, robustness, and usability scores is provided in Figure 14 and Figure 15. Some observations can be made regarding flexibility and usability scores and future proof and usability scores. First, two out of five software with a high future proof rating received a high usability rating. This might indicate that a high future proof rating is not necessarily related to high usability. Second, the three software with a medium future proof rating also received a medium usability rating. The other software tool with a medium future proof score did not have a usability score. Third, five out of seven software tools that scored high for flexibility scored medium for usability. Notably, no software with a medium future proof or flexibility score received a high usability score.

Relating to the high flexibility and medium usability scores, the high flexibility of the software has been mentioned by interviewees as a possible factor of increased complexity for users. The implementer from Software 7 also explains this for a project he has implemented as follows "*… it's sometimes because the clients customise it heavily, and then it makes it more complex as well*.". The same applied to the implementer from Software 3, that explained how customers sometimes make it complex for themselves as they request a vast number of changes.

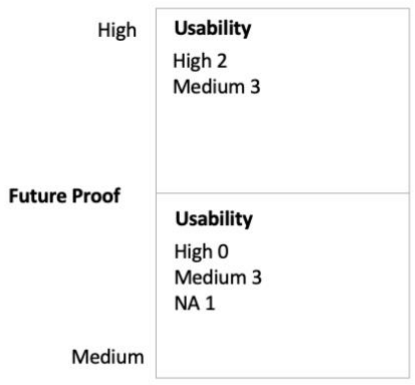


Figure 15 Overview of the relationship between future proof and usability

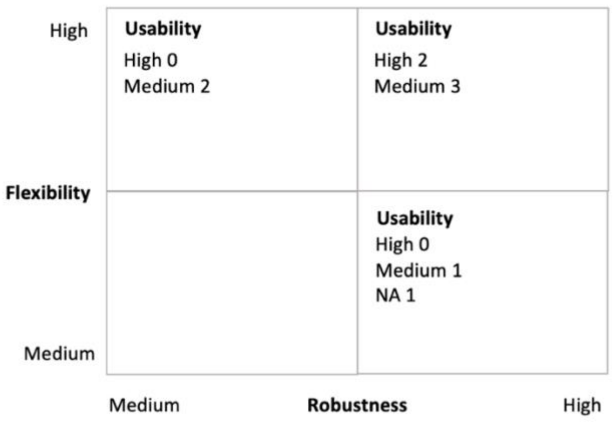


Figure 14 Overview of the relationship between flexibility, robustness and usability

As mentioned before, ESG is fairly data-driven, and the amount of data that needs to be reported on for new upcoming regulations is extensive. Interviewees have mentioned how a vast amount of data fields or the complexity of the software impacts the ease of use for end-users. This is also explained in the following quote by the vendor from Software 5 "*So a user will go this system is hard to use, but it's not hard because we did that, but it's because the customer wants to fill in 400 fields*". Therefore, a specific ESG aspect might be linked to the usability of the software as well as many different data points that need to be added to ESG software tools.

The two software that did score high for usability were relatively mature. For one of the software, the ESG solution is only added as an add-on. Consequently, users were already familiar with the software, leading to a higher usability score, as explained by the implementer of Software 1. *"… as I said, we use customers who are already 'Software 1' customers, so they already use the technology and it's just the same interface. It's the same logic. Creating a report is the same thing as they do today for the modules, so it's very easy."* Usability of the other software was scored as high due to its maturity resulting in an easy-to-use and intuitive software, as explained by the second implementer of Software 2. *"… when we say mature, one of the reasons is how fault programme it is, how easy it is to use, how much-varied functionality it gives, how much we can tweak the existing structure to the advantage of any particular use case, and it normally takes bots in most of if not all of them in that sense, yeah.".*

This analysis will also reflect robustness scores to examine if there was a potential relation between robustness and usability, as it is part of future proof as well. A medium score for robustness also resulted in a medium score for usability, but high robustness only resulted in high usability in two out of five cases (see Figure 15). One of the causes of this was that due to robustness problems, the performance of the software was lacking, which impacted the whole user experience. The implementer from Software 6 explained, *"Or there should be functional and technical consultants which, when we are implementing a system, will need to kind of set boundaries, OK. That if you go beyond that, you're impacting the usability and performance of a system to a point where it might not be liked by the end users. So, they need to probably define the boundaries".*

The other software that scored medium for robustness reflects how usability scores are fairly lower. This was explained in the interview as an interplay between high customisation, which will lead to a more complex software to use and also results in robustness problems as the performance of the software is not working as it is intended to work. According to the implementer from Software 5, this has to do with the fact that "*the clients customise it heavily, and then it makes it more complex as well*." And later on, explains that less customisation is better for performance.

Hence, based on the results from this thesis, the second proposition is not supported. No software scored low for either flexibility or future proof and therefore, this relation could not be explored. Furthermore, medium flexibility resulted in one instance of medium usability, but this was also the only case with a medium flexibility score and a present usability score. Notably, software that scored high for flexibility mainly received medium usability scores and that none of the software with medium future proof or flexibility scores received a high usability rating. This might indicate that there is a possible trade-off between high flexibility, which can result in lower usability scores.

5 DISCUSSION & CONCLUSION

This chapter discusses the results of the study and provides possible explanations why both propositions were not supported. Additionally, the limitations of this study with accompanying recommendations for future research are discussed, followed by contributions of this study to the literature. At the end of this chapter, the conclusion is provided, which answers the posed research question and eventually, the managerial implications of this study are provided.

5.1 Discussion

5.1.1 Future proof

Due to changing regulations for organisations and evolving business needs to become more sustainable, organisations are shifting to software tools to report on their ESG efforts. ESG software needs to be future proof to cope with these changing regulations and business needs (Said, et al., 2015; Helbig, et al., 2021; Pee, et al., 2021). Previous literature did not cover to what extent ESG software tools are future proof and how this can be measured. Therefore, this thesis aimed to study to what extent ESG software tools are future proof and which factors relate to this. By operationalising future proof in the concept's flexibility and robustness in the context of ESG reporting software, the concept of future proof has been made measurable. Even though both propositions were not supported, various results emerged relating to the future proof factors and additional insights that might relate to these future proof ratings.

First of all, it is interesting to note that all participating software tools scored either medium or high for the concept of future proof, and no software received a low score. This means that the overall score for future proof was relatively high. A possible explanation for this is that all software tools included in this study offer renowned ESG reporting solutions, which have been implemented in many larger organisations and were preselected by an external research platform. So, even though the proposed model to measure future proof seemed to be successful, the sampling approach might have had an impact on the eventual future proof outcomes.

The specific context in which this study has been performed might also have influenced the eventual future proof scores of these software tools. ESG is a fairly data-driven problem, and data is coming from a vast number of places. In addition, it was mentioned how software tools are still figuring out how to perfectly design their tools for ESG data. This also relates to the findings of a study that suggested that ESG data and reporting are not as mature as financial reporting (Littan, 2019).

5.1.2 Software design features

It seemed that three main approaches on how to tackle ESG data complexity were formed. First, one group of software tools approached more of a generalist approach allowing end-users to modify most aspects of the software. Second, software tools that were specifically designed emerged, which, for a large part, allowed modification of the software. The last group are very specific niche software which focuses on detailed calculations. A different level of flexibility was perceived for more niche-focused software that aimed to tackle the data collection of ESG. The two software tools with a more niche focus towards ESG scored lower for flexibility, as they aimed to offer software made for all users, including complex calculations and detailed sustainability metrics.

5.1.2.1 Future proof and architecture

Previous literature studies suggested that architecture relates to future proof information systems (Bass, et al., 2012; Furrer, 2019). The specific architecture styles of serverless, microservice and service-oriented architecture were considered to relate to robust and flexible software (Newman, 2015; Balalaie, et al., 2016; Auer, et al., 2021; Hustad & Olsen, 2021; Slamaa, et al., 2021), whereas a monolithic architecture was considered to be less related to flexible and robust software (Götz, et al., 2018). Hence the following proposition was formed: *Serverless, Microservice, and Service-oriented architecture styles will likely relate to future proof software systems, whereas monolithic architecture will likely not relate to future proof software systems.*

This proposition was not supported by the findings of this study, as serverless, microservice or service-oriented architectures did not necessarily relate to future proof software tools. These three architectures were related to software tools with medium and high outcomes for future proof. Furthermore, it was expected that a monolithic architecture would likely not relate to future proof software. The software that included a monolithic architecture scored medium for future proof. However, the solution used a combination of architectures, making it unclear to distinguish the influence of the monolithic architecture. Therefore, no clear patterns emerged between future proof software and specific architecture styles. These findings do not align with previous literature that did indicate that these architectural styles led to more flexible and robust or less flexible and robust software systems. This can possibly be explained by the theory that was used. Most literature used was largely based on general enterprise software (Newman, 2015; Balalaie, et al., 2016; Götz, et al., 2018; Auer, et al., 2021; Hustad & Olsen, 2021; Slamaa, et al., 2021) and has not necessarily been specified towards ESG reporting software. This thesis specifically focused on ESG reporting solutions. Two possible explanations for the difference between the literature and the findings in this study might relate to the selected quality sub-characteristics, and other design factors that influence the degree to which ESG reporting software tools are future proof.

According to findings from past literature, specific quality characteristics were identified for sustainability software, and in this study, these were linked to the various architecture styles. Eventually, the quality sub-characteristics analysability, installability, modularity, interoperability, replaceability and reusability were considered as these seemed to relate to the considered architecture styles. As good as all software covered all the quality sub-characteristics that were selected for this study. This finding was expected as every software tool either had a cloud microservice or service-oriented architecture in combination with another architecture type. Therefore, these findings align with existing literature that highlighted how these specific quality characteristics applied to microservice and service-oriented architecture styles (Fink & Neumann, 2009; Lenhard, et al., 2013; Newman, 2015; Balalaie, et al., 2016; Haoues, et al., 2017; Auer, et al., 2021; Slamaa, et al., 2021). Furrer (2019) identified that quality characteristics are essential for future proof software. These quality characteristics were used as an additional factor to form the first proposition and to determine differences in quality for the various architecture types. It also seemed that quality characteristics do not necessarily relate to the extent to which an ESG reporting software tool is future proof, as software that included all quality sub-characteristics also received medium future proof scores.

Thus, it seems that architecture style and quality sub-characteristics do not necessarily relate to the extent to which an ESG software tool is future proof. Even though the proposition was not supported, the findings did suggest a variation between the future proof scores for the participating ESG software tools and provided insights into other factors that might possibly explain these differences. The cloud service methods, coding methods, and offering were related concepts to the architecture that, in these nine cases, might be related to the robustness and flexibility of this software.

5.1.2.2 Additional features

When focusing on cloud service methods, the findings suggested that software offered as a platform as a service (PaaS) is possibly related to high future proof scores. The increased speed at which applications can be developed with PaaS and the benefit of having all ESG-related data in one platform might result in a higher future proof rating. Especially in the context of ESG, data handling is an important aspect, and by hosting multiple business applications on one platform, ESG-related data is already available on the platform and can be linked more easily for reporting purposes. These findings align with other literature suggesting how PaaS leads to scalable and robust enterprise software (Braubach, et al., 2011). No relationship has been observed for software as a service (SaaS).

Moreover, in this study, it was indicated that a low code, no code method possibly positively relates to the flexibility of ESG software. The four cases that included low code, no code method all appeared to score high for flexibility. Low code, no code methods improve the speed by which changes are made due to limited coding and allows users to control more software aspects, which might relate to improved flexibility for this software. These findings are also in line with a study performed by Rafi et al. (2022), which argues that low code can speed up software development to meet customer needs. De Vries & Stam (2019) also argue that low code, no code can contribute to more flexible and adaptable software. Despite the small sample size, the findings suggest that flexibility can be increased through low code, no code software. These observations align with other scientific research that acknowledges the influence of low code, no code software on the flexibility of ESG reporting software and further strengthens the existing literature on this matter.

Related to the offering, the study also found that both off-the-shelf and custom-developed software consistently received high flexibility ratings and off-the-shelf high robustness scores. However, the software, which only offered an off-the-shelf ESG solution, also scored high for flexibility and software tools that allowed modification or customisation for robustness. This suggests that the presence of a modifiable off-the-shelf or custom-development offering may relate to high flexibility, but the absence of modifiable off-the-shelf or custom-developed software does not necessarily lead to inflexible software. In terms of off-the-shelf, the presence of an off-the-shelf offering may positively relate to high robustness, but the absence does not necessarily lead to low robustness. These findings are also in line with literature that highlighted how organisations choose custom-developed or modifiable off-the-shelf software for the flexibility, adaptability and configurability of the two options (Mousavidin & Silva, 2017; Shahzad, et al., 2017; Singh & Pekkola, 2021). Other literature highlights the robustness and reliability of off-the-shelf software as these software tools are tested and pre-made (Hutchinson, et al., 2003; Spurrier & Topi, 2017).

5.1.3 Usability

Besides architecture, usability has been identified in other studies as an important quality characteristic for ESG reporting software (Saher, et al., 2020). Most software scored medium for usability, and only two software scored high for usability. None of the tools received a low score for usability. Again, this could relate to the fact that most software tools are relatively renowned and would not be used by as many organisations if this aspect was lacking. However, it is interesting to note that exclusively two out of nine software tools scored high for usability.

A possible relation between flexibility and usability has been identified in the literature. Previous literature mentioned how inflexible software leads to usability problems and affects the user's experience with the software tool (Mahrin, et al., 2008; Li & Nielsen, 2019; Rakovic, et al., 2020). Hence the following proposition was proposed: *A low flexibility rating and, thus, a low future-proof rating will likely relate to low usability.*

This proposition was not supported as none of the participating software vendors scored low for flexibility. Three interesting findings did emerge. First, two out of seven software tools that scored high for flexibility and two out of five that scored high for future proof also scored high for usability. However, this did not indicate that all highly flexible tools related to high usability scores. Second, software that scored high on future proof and flexibility in most cases received medium usability scores. Lastly, one software that scored medium for flexibility also scored medium for usability, but as this was only one case, no strong assumption can be made.

The first finding could suggest that high flexibility and high future proof ratings could result in high usability. This finding aligns with existing literature that highlights how highly flexible software increases the user-friendliness of software systems (Palanisamy, 2012; Li & Nielsen, 2019). In addition, interviewees added how highly flexible software tools, in some cases, allow users to modify their users' interfaces and adjust components to suit the organisation's business processes. In this way, a software tool would align with an organisation's internal business processes, which eases the use of this software.

The second finding highlighted another possible impact of flexibility on usability. High flexibility scores are related in multiple instances to medium usability scores. This finding would align with flexibility challenges that were reported in existing studies. These studies suggested a trade-off between flexibility and usability. Meaning that highly flexible software tools are likely to be more complex, which could lead to lower usability ratings (Lidwell, et al., 2010). High complexity in software is perceived by users as difficult to learn and understand. In this study, five out of seven software that scored high for flexibility scored medium for usability. This might indicate that highly flexible ESG reporting software scored lower for usability. Interviewees also identified how the increasing complexity of the software increased with high flexibility, and this eventually impacted the user's experience.

Another explanation for possible differences in the data and a possible reason why only two software scored high for usability could be explained by the specific ESG context. ESG is fairly data-driven, and a fast amount of data fields need to be added coming from different data sources and other systems. This further influences the experience of eventual users of these tools. It has been noted by interviewees how an extensive amount of data fields leads to more complex software. In the case of ESG reporting, such as the CSRD, end-users will need to report on at least 1000 data points increasing complexity. Thus, ESG software and the amount of data that needs to be handled can possibly lead to higher complexity of software tools and lower usability ratings.

While the proposition suggesting a relationship between flexibility, future proof and usability ratings was not supported in this study, several interesting findings and potential explanations for the differences in results emerged.

5.2 Limitations and future research

Even though valuable insights can be derived from this thesis, there are various limitations that should be considered and accompanying future research topics are discussed.

The research design of this study was primarily a qualitative design, and therefore, it was not expected that a large amount of ESG software cases would participate. However, the specific sampling of this study might have led to the low variation in future proof scores. The participating ESG software tools were preselected and mostly renowned software with a large customer base. None of the software received a low score for either future proof or usability. Therefore, these cases might not completely represent the entire ESG reporting software tool market. Future research could include a larger sample size, including renowned and less renowned software tools, to ensure a proper representation of the full market. A consequence of this could relate to complexity in generalisation, as a larger heterogenous variation could lead to even more conditions that influence the eventual future proof or usability ratings.

Furthermore, it can be questioned if software tools that score low on future proof would still be present as they would likely be overruled by other software tools.

Secondly, the study is for one part based on interviews. This introduces the possibility of biased or incomplete information, as the participants might provide favourable or selective information about their software solutions. To avoid this, several data sources were used to provide information about the software solutions, including an external research platform and the researcher's judgement. However, not all cases included a software vendor, implementer or external research. As there was quite some consistency between the various sources used, it is not expected that interviews with these additional vendors and implementers would impact the eventual outcomes.

Thirdly, not all interviewees could answer all questions due to knowledge limitations. Certain vendors or implementers did not implement a new regulation yet or did not experience a lot of bugs relating to a change in the software. Thus, certain software tools only covered a part of the structured interview questions, leading to a difference in the scope covered for each software tool. An average of the scores that could be given was taken, but fewer subjects were included than for other cases. The argumentation behind the scores was considered even more valuable for these specific cases.

Finally, although the factors were based on literature, some external factors were mentioned during the interviews that related to flexibility, robustness and usability of the software. In certain cases, a pattern emerged for these factors and in other cases, the interviewees referred to these factors, but no pattern was visible in the current data. Future research could focus on a configurational approach to see if a combination of these factors would lead to specific findings relating to flexibility, robustness or usability. In this thesis, all factors were considered separately and at first glance, there did not seem to be one specific combination of factors that led to high or medium scores. A larger number of cases in combination with a configurational approach could examine if there would be a certain configuration that does result in high or medium scores.

Various suggestions for future research will be provided relating to the three factors that did seem to possibly relate to either future proof in general, flexibility or robustness. First, in the case of the various cloud service methods, it seemed that PaaS related to high future proof ESG software tools in this study. As only two cases were included in this study that used a PaaS cloud service, it would be interesting for future research to explore if these findings relate to more ESG software tools that use a PaaS cloud service. Second, software that used low code, no code as a coding method scored high for flexibility in all four cases. It would be interesting to see if this would relate to a study with a larger number of cases. Eventually, the offering of the software possibly related to flexibility, modifiable and custom-developed software scored higher for flexibility. The offering also seemed to be related to robustness as off-the-shelf software related to higher robustness scores. Future research could, with the use of a quantitative study, examine if there are significant relations between offering, flexibility and robustness.

Other factors that did not show a clear pattern in this study were the heritage of the ESG software, the cloud deployment model, the tenancy of the software and the maturity of the software in terms of the time it had been on the market. Interviewees suggested that there were possible influences of these factors on either flexibility or robustness. Future research could further examine with a configurational approach whether there are configurations and the interplay among these factors that might affect flexibility or robustness. A configurational approach could thus help analyse how the combination of various factors influences the desired extent to which software tools are future proof.

5.3 Contribution to literature

The study contributes to the literature on ESG reporting software by exploring the concept of future proof in the specific context of ESG reporting software. Previous literature emphasised the importance of future-proof information systems, but this had not been studied in the specific ESG context. This study fills that gap by focusing on the extent to which current ESG reporting software tools are future proof and which factors relate to this.

Previous literature highlighted how architecture as a design feature is an important aspect that might influence the extent to which an information system is future proof. The findings showed a difference between previous assumptions regarding the relationship between architecture styles and future-proof software. The study did not find clear patterns or support for the proposition that specific architecture styles directly relate to future-proof software. The findings indicate that more design features might relate to future proof ESG reporting software. This study highlights which factors could contribute to a more flexible or robust software and that architecture on its own will not reach this for ESG reporting software. Given that this research area was previously underexplored, the results of this thesis can potentially contribute to future research by enhancing our understanding of the conditions in which ESG reporting software can be flexible and robust and thus future proof.

Besides the future proof indication in relation to the design or architecture of the software, usability was considered to be an important factor as well. Previous literature discussed various relations between usability and flexibility. On the one hand, it was mentioned how low flexibility would relate to low usability and, on the other hand, how highly flexible software might relate to lower usability due to the complexity of the software. The findings seem to support the latter, as most software with high flexibility scored medium in terms of usability. It must be noted that this was not the case for all software solutions, as some scored high for both flexibility and usability, indicating that even though there might be a trade-off, this does not relate to all software solutions. In addition, the complexity of ESG data collection adds another component that could influence the eventual usability of the software tools. Therefore, the contributions of this study might indicate that the relation between flexibility and usability is more complex than what was initially expected, especially in the context of ESG reporting software.

The study's findings contribute to the existing literature by uncovering the future proof degree within the context of ESG reporting software. This highlights the need for further research to examine the differences between general enterprise software and ESG-specific software, as well as the interplay between design features, usability, and future-proof ratings. Overall, the study enhances the understanding of possible factors that relate to future proof ESG reporting software, which opens avenues for future research in this specific domain.

5.4 Conclusion

This study aimed to understand how various software architectures and system quality characteristics relate to the flexibility and robustness of ESG reporting systems. A comparative case study analysis was used to compare the various software and identify patterns in the data. Nine ESG software solutions were included in the thesis. For each case, a combination of desk research and interviews was used. In total, five different desk research approaches have been used, and fifteen interviews have been conducted to answer the following research question:

How do system architecture and usability relate to ESG reporting software system's ability to be future proof?

This study's findings suggest that architecture and the usability of software are not necessarily related to the extent to which a software system is future proof.

It seemed that architecture on its own does not necessarily relate to future proof scores. Especially since architecture styles were expected to relate to highly flexible and robust software tools, and thus future proof software tools did not necessarily relate to high future proof ESG reporting software. Various other factors were identified that could possibly relate to higher future proof ratings, such as the cloud service, coding method and offering of the software. To start with, the specific cloud service PaaS was considered to possibly relate to future proof software as it allows fast modification of applications on the platform. Second, software including low code, no code methods seemed to enhance flexibility for all software tools that included this coding method. The same accounted for modifiable off-the-shelf and custom-developed software. These offering types possibly had a positive influence on the flexibility of the software, whereas the offering type off-the-shelf is likely related to higher robustness. Furthermore, it was observed that more specified ESG software scored a lower flexibility rating due to the extensiveness and lack of customisation of these solutions.

Relating to usability, the findings did suggest that there might be a possible trade-off between flexibility and usability. Most software tools that received high flexibility scores received medium usability scores due to the increased complexity of the software. In addition, high future proof scores were also related to high usability in other instances. Therefore, it might be possible that future proof and usability are related in two ways. The specific ESG context seemed to also have an influence on the complexity of the software tools due to the complexity of data collection and eventually impacted the usability of the system.

Thus, the findings suggest that future proof software is not solely related to architecture or usability, but many more factors influence the eventual extent to which ESG reporting software tools are future proof.

5.5 Managerial implications

The findings of this study suggest practical implications for software vendors, system implementers and end-users. Multiple design factors seem to relate to the extent to which an ESG software tool is future proof, and therefore there is not a single blueprint for the most preferred software design. However, some factors influenced the future proof ratings and should be considered.

To start with the implications for vendors. The research highlighted possible implications for improvements of the current designs of ESG reporting software. In terms of the concept of future proof, it seemed that the specific cloud service type PaaS related to highly future proof software. In addition, the high flexibility of a software tool related to custom-developed or modifiable software and to software that offered low code, no code coding methods. In terms of robustness, high robustness was related to off-the-shelf software. Some suggestions that were provided relating to robustness issues included increased testing of software, service locations in every content to continuously work on bugs and setting boundaries in the number of modifications that users can make. These are some characteristics that vendors can take into consideration when designing for future proof ESG software tools. It must be mentioned that the absence of these factors did not necessarily lead to lower flexibility or future proof ratings. The usability of the software can be compromised in highly customisable and complex software. Some suggestions that have been provided to improve usability include increased training sessions for users and increased participation of users as software vendors have limited business knowledge.

Besides vendors, there are also various implications for implementers. The specific characteristics that possibly enhance future proof ESG software can be taken into account by implementers when selecting and comparing vendors for clients. Implementers usually act for their clients, and the first step for an implementer would be to consider the end-user’s objectives. An implementer may opt for less future proof, but more detailed ESG software if that matches the client’s objectives or a less future proof but more complex software. An example of an implication for implementers could be a decision tree which takes into account the various design features discussed in this study to eventually match an ESG software tool to various clients.

End-users, in this sense, relate to the clients of implementers and organisations that need to report on specific ESG principles. These organisations will likely be large-scale and subjected to mandatory reporting. First of all, end-users can use the findings of this study on the extent to which the current ESG software tools are future proof and factors that influence this when selecting a new software vendor. In addition, end-users should consider that in cases of highly flexible software, a lot of functionalities will need to be configured, and they will have to weigh up the extent to which they have the in-house knowledge to make adjustments to a software solution. In cases of less technical knowledge on a user's end, it might be more interesting to hire a system implementer or opt for software with low code, no code methods to adjust the software systems.

6 LITERATURE REFERENCES

Al-Debagy, O., & Martinek, P. (2018). A comparative review of microservices and monolithic architectures. *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)* (pp. 000149-000154). Budapest: IEEE.

Almagtome, A., Khaghaany, M., & Önce, S. (2020). Corporate governance quality, stakeholders' pressure, and sustainable development: an integrated approach. *International Journal of Mathematical Engineering and Management Sciences, 5*(6), 1077-1090.

Auer, F., Lenarduzzi, V., Felderer, M., & Taibi, D. (2021). From monolithic systems to microservices: An assessment framework. *Information and Software Technology, 137*, 106600.

Bakarich, K. M., Castonguay, J. J., & O'Brien, P. E. (2020). The use of blockchains to enhance sustainability reporting and assurance. *Accounting Perspectives, 19*(4), 389-412.

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables devops: migration to a cloud-native architecture. *IEEE Software, 33*(3), 42-52.

Barber, F., & Salido, M. A. (2015). Robustness, stability, recoverability, and reliability in constraint satisfaction problems. *Knowledge and Information Systems, 44*(3), 719-734.

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd ed.). Boston: Addison-Wesley.

Bauer, W., & Maurer, M. (2011). Future-proof interfaces: Systematic identification and analysis. *13th International Dependency and Structure Modelling Conference, Dsm'11*, (pp. 89-101). Cambridge, Massachusetts.

Bhargava, H. K., & Gangwar, M. (2016). "Pay as you go" or" all you can eat"? pricing methods for computing and information services. *49th Hawaii International Conference on System Sciences (HICSS)* (pp. 5239-5248). IEEE.

Braubach, L., Pokahr, A., & Jander, K. (2011). Jadexcloud-an infrastructure for enterprise cloud applications. *Multiagent System Technologies: 9th German Conference, MATES* (pp. 3-15). Berlin, Germany: Springer.

Bryman, A. (2012). The nature of qualitative research. In A. Bryman, *Social research methods* (4th ed., pp. 379-414). Oxford: Oxford University Press.

Calero, C., Bertoa, M. F., & Moraga, M. Á. (2013). A systematic literature review for software sustainability measures. *2013 2nd international workshop on green and sustainable software (GREENS)* (pp. 46-53). San Francisco: IEEE.

Calero, C., Moraga, M. Á., Bertoa, M. F., & Duboc, L. (2014). Quality in use and software greenability. *Proceedings of CEUR Workshop* (pp. 28-36). CEUR.

Carney, D., & Leng, F. (2000). What do you mean by COTS? Finally, a useful answer. *IEEE software, 17*(2), 83-86.

Cerny, T., Donahoo, M. J., & Trnka, M. (2018). Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review, 17*(4), 29-45.

Chelawat, H., & Trivedi, I. V. (2016). The business value of ESG performance: the Indian context. *Asian journal of business ethics, 5*(1), 195-210.

Chen, R. S., Sun, C. M., Helms, M. M., & Jih, W. J. (2009). Factors influencing information system flexibility: an interpretive flexibility perspective. *International Journal of Enterprise Information Systems (IJEIS), 5*(1), 32-43.

Chofreh, A. G., Goni, F. A., Klemeš, J. J., Malik, M. N., & Khan, H. H. (2020). Development of guidelines for the implementation of sustainable enterprise resource planning systems. *Journal of Cleaner Production, 244*, 118655.

Chofreh, A. G., Goni, F. A., Shaharoun, A. M., Ismail, S., & Klemeš, J. J. (2014). Sustainable enterprise resource planning: imperatives and research directions. *Journal of Cleaner Production, 71*, 139-147.

Commission of the European Communities. (2001). *Promoting a European framework for corporate social responsibilities.* Brussels: COM.

Cruz, C. A., & Matos, F. (2023). ESG maturity: a software framework for the challenges of ESG data in investment. *Sustainability, 15*(3), 2610.

De Vries, B., & Stam, D. (2019). Low-Code and the road to sustainable software. *Compact, 2*, 30-39.

Druzhaev, A. A., Isaev, D. V., & Ogurechnikov, E. V. (2019). Principles of managing development of EPM systems. *Бизнес-информатика, 13*(2), 73-83.

Du, S., Bhattacharya, C. B., & Sen, S. (2010). Maximizing business returns to corporate social responsibility (CSR): the role of CSR communication. *International journal of management reviews, 12*(1), 8-19.

Dvořák, O., Pergl, R., & Kroha, P. (2017). Tackling the flexibility-usability trade-off in component-based software development. *Recent Advances in Information Systems and Technologies, 1*(5), 861-871.

Enghaug, M. M., & Hallan, R. L. (2022). *The demand for specialised ESG reporting software in the Norwegian ESG landscape (Bachelor's thesis).* Trondheim: NTNU.

European Commission. (2021a). *EU taxonomy, corporate sustainability reporting, sustainability preferences and fiduciary duties: Directing finance towards the European Green Deal.* Brussels: European Commission.

European Commission. (2021b). *Proposal for a directive of the European Parliament and of the council amending Directive 2013/34/EU, Directive 2004/109/EC, Directive 2006/43/EC and Regulation (EU) No 537/2014, as regards corporate sustainability reporting.* Brussels: European Comission. Opgeroepen op December 5, 2022, van https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021PC0189

Fernandez-Feijoo, B., Romero, S., & Ruiz, S. (2014). Effect of stakeholders' pressure on transparency of sustainability reports within the GRI framework. *Journal of business ethics, 122*(1), 53-63.

Fink, L., & Neumann, S. (2009). Exploring the perceived business value of the flexibility enabled by information technology infrastructure. *Information & Management, 46*(2), 90-99.

Fortuna, F., Testarmata, S., Sergiacomi, S., & Ciaburri, M. (2020). Mandatory disclosure of non-financial information: a structured literature review. *Accounting, Accountability and Society: Trends and Perspectives in Reporting, Management and Governance for Sustainability*, 95-128.

Furrer, F. J. (2019). *Future-proof software systems: A sustainable evolution strategy.* Cham: Springer.

Garg, S. K., Versteeg, S., & Buyya, R. (2011). Smicloud: a framework for comparing and ranking cloud services. *2011 Fourth IEEE International Conference on Utility and Cloud Computing* (pp. 210-218). Melbourne: IEEE.

Gebauer, J., & Schober, F. (2006). Information system flexibility and the cost efficiency of business processes. *Journal of the association for information systems, 7*(3), 8.

Gillan, S. L., Koch, A., & Starks, L. T. (2021). Firms and social responsibility: a review of ESG and CSR research in corporate finance. *Journal of Corporate Finance, 66*, 101889.

Götz, B., Schel, D., Bauer, D., Henkel, C., Einberger, P., & Bauernhansl, T. (2018). Challenges of production microservices. *Procedia CIRP, 67*, 67-172.

Green, P., Robb, A., & Rohde, F. H. (2014). A model for assessing information systems success and its application to e-logistics tracking systems. *Pacific Asia Journal of the Association for Information Systems, 6*(4), 3.

Haoues, M., Sellami, A., Ben-Abdallah, H., & Cheikhi, L. (2017). A guideline for software architecture selection based on ISO 25010 quality related characteristics. *International Journal of System Assurance Engineering and Management, 8*(2), 886-909.

Hasselbring, W., & Steinacker, G. (2017). Microservice architectures for scalability, agility and reliability in e-commerce. *Proceedings International Conference on Software Architecture Workshops (ICSAW)* (pp. 243-246). Gothenburg: IEEE.

Helbig, R., von Höveling, S., Solsbach, A., & Marx Gómez, J. (2021). Strategic analysis of providing corporate sustainability open data. *Intelligent Systems in Accounting, Finance and Management, 28*(3), 195-214.

Hilpert, H., Kranz, J., & Schumann, M. (2014). An Information system design theory for green information systems for sustainability reporting-integrating theory with evidence from multiple case studies. *Proceedings of the European Conference on Information Systems (ECIS) 2014.* Tel Aviv: Association for Information Systems.

Hoang, G., Molla, A., & Poon, P. L. (2016). How do environmental enterprise systems contribute to substainability value? A practitioner-oriented framework. Wollongong, NSW: Australasian Conference on Information Systems.

Hoang, G., Molla, A., & Poon, P. L. (2017). An exploratory study into the use and value of environmental enterprise systems. *Australasian Conference on Information Systems* (p. 68). Hobart: ACIS.

Hoang, G., Molla, A., & Poon, P. L. (2019). Factors influencing the adoption of environmental enterprise systems. *PACIS 2019 Proceedings*, (p. 196).

Hustad, E., & Olsen, D. H. (2021). Creating a sustainable digital infrastructure: the role of service-oriented architecture. *Procedia Computer Science, 181*, 597-604.

Hutchinson, J. E., Kotonya, G., & Sawyer, P. (2003). Understanding the impact of change in COTS-based systems. *Software Engineering Research and Practice*, 752-760.

IDC. (2023). *International Data Corporation (IDC)*. Opgeroepen op January 6, 2023, van IDC: https://www.idc.com/about

ISO. (2011). *ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.* International Organization for Standardization.

ISO/ IEC / IEEE. (2022). *ISO/IEC/IEEE 42010:2022 Software, systems and enterprise — Architecture description.* ISO. Opgeroepen op January 6, 2023, van https://www.iso.org/standard/74393.html

Jacome, L., Byrd, T. A., & Byrd, L. W. (2011). An examination of information systems flexibility. *International Journal of Information Processing and Management, 2*(2), 69-77.

Jamous, N., Alwafaie, R., & Dahma, M. A. (2012). Corporate environmental management information systems (CEMIS)-sustainability reporting tools for SMEs. *EnviroInfo*, 657-664.

Jiménez-Ramírez, A., Weber, B., Barba, I., & Del Valle, C. (2015). Generating optimized configurable business process models in scenarios subject to uncertainty. *Information and Software Technology, 57*, 571-594.

Kavis, M. J. (2014). Cloud service models. In *Architecting the cloud* (pp. 13-22). Hoboken: John Wiley & Sons.

Knoll, K., & Jarvenpaa, S. L. (1994). Information technology alignment or “fit” in highly turbulent environments: the concept of flexibility. *Proceedings of the 1994 computer personnel research conference on Reinventing IS: managing information in changing organizations* (pp. 1-14). Virginia: SIGCPR'94.

Koçak, S. A., Alptekin, G. I., & Bener, A. (2014). Evaluation of software product quality attributes and environmental attributes using ANP decision framework. *Proceedings of CEUR Workshop* (pp. 37-44). CEUR.

Koziolek, H. (2011). Sustainability evaluation of software architectures: a systematic review. *Proceedings of the joint ACM SIGSOFT conference--QoSA and ACM SIGSOFT symposium--ISARCS on Quality of software architectures--QoSA and architecting critical systems--ISARCS* (pp. 3-12). Boulder: QoSA-ISARCS.

Lakhai, V., & Bachynskyy, R. (2021). Investigation of serverless architecture. *Advances in Cyber-Physical Systems, 6*(2), 134-139.

Lenhard, J., Harrer, S., & Wirtz, G. (2013). Measuring the installability of service orchestrations using the square method. *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications* (pp. 118-125). Koloa: IEEE.

Li, M., & Nielsen, P. (2019). Making usable generic software. A matter of global or local design? *Tenth Scandinavian Conference on Information Systems (SCIS2019).* Nokia, Finland.

Lidwell, W., Holden, K., & Butler, J. (2010). *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design.* Rockport Pub.

Littan, S. (2019). The COSO internal control framework and sustainability reporting. *The CPA Journal, 89*(7), 22-26.

Loeser, F., Recker, J., Brocke, J. V., Molla, A., & Zarnekow, R. (2017). How IT executives create organizational benefits by translating environmental strategies into Green IS initiatives. *Information Systems Journal, 27*(4), 503-553.

Losavio, F., Chirinos, L., Lévy, N., & Ramdane-Cherif, A. (2003). Quality characteristics for software. *Journal of Object Technology, 2*(2), 133-150.

Mahinda, E., & Whitworth, B. (2004). Evaluating flexibility and reliability in emergency response information systems. *Proceedings ISCRAM2004, 93*, 93-98.

Mahrin, M. N., Carrington, D., & Strooper, P. (2008). Investigating factors affecting the usability of software process descriptions. *Making Globally Distributed Software Development a Success Story: International Conference on Software Process* (pp. 222-233). Leipzig, Germany: Springer.

Martinsons, M., Davison, R., & Tse, D. (1999). The balanced scorecard: a foundation for the strategic. *Decision Support Systems, 25*, 71–88.

McEwan, M., Shi, Y., & Van Toorn, C. (2021). Business analytics (BA)-powered transformation for environmental sustainability in organisations: A dynamic capabilities perspective. *ACIS 2021 Proceedings* (p. 88). Sydney: ACIS.

Mietzner, R., Unger, T., Titze, R., & Leymann, F. (2009). Combining different multi-tenancy patterns in service-oriented applications. *IEEE International Enterprise Distributed Object Computing Conference* (pp. 131-140). IEEE.

Miles, M. B., Huberman, A. M., & Saldaña, J. (2014). Designing matrix and network displays. In *Qualitative data analysis - A method sourcebook* (3rd ed., pp. 107-120). Sage Publications.

Mion, G., & Loza Adaui, C. R. (2019). Mandatory nonfinancial disclosure and its consequences on the sustainability reporting quality of Italian and German companies. *Sustainability, 11*(7), 4612.

Mitropoulos, S., & Douligeris, C. (2011). The impact of new service oriented architectures technologies in the new global market oriented enterprises. *International Journal of Applied Systemic Studies, 4*(1/2), 106-120.

Mousavidin, E., & Silva, L. (2017). Theorizing the configuration of modifiable off-the-shelf software. *Information Technology & People, 30*(4), 887-909.

Muntes Mulero, V., Matthews, P., Omerovic, A., & Gunka, A. (2013). Eliciting risk, quality and cost aspects in multi-cloud environments. *Proceedings Fourth International Conference on Cloud Computing, Grids, and Virtualization (CLOUD COMPUTING 2013)* (pp. 238-243). IARIA.

Newman, S. (2015). Microservice. In *Building microservices* (pp. 1-12). Sebastopol: O'Reilly Media, Inc.

O'Meara, W., & Lennon, R. G. (2020). Serverless computing security: Protecting application logic. *31st Irish Signals and Systems Conference (ISSC)* (pp. 1-5). IEEE.

Odun-Ayo, I., Ananya, M., Agono, F., & Goddy-Worlu, R. (2018). Cloud computing architecture: a critical analysis. *18th international conference on computational science and applications (ICCSA)* (pp. 1-7). Melbourne: IEEE.

Oh, S. H., La, H. J., & Kim, S. D. (2011). A reusability evaluation suite for cloud services. *2011 IEEE 8th International Conference on e-Business Engineering* (pp. 111-118). Beijing: IEEE.

Oudshoorn, I. (2004). *Development of packaged software.* Amsterdam: Vrije Universiteit Amsterdam.

Palanisamy, R. (2012). Building information systems flexibility in SAP–LAP framework: a case study evidence from SME sector. *Global Journal of Flexible Systems Management, 13*, 57-74.

Palanisamy, R., & Boyle, T. (2010). A framework for managing enterprise systems flexibility. *Journal of E-Technology, 1*(3), 131-139.

Pan, S. L., Carter, L., Tim, Y., & Sandeep, M. S. (2022). Digital sustainability, climate change, and information systems solutions: Opportunities for future research. *International Journal of Information Management, 63*, 102444.

Pan, Y., & Hu, N. (2014). Research on dependability of cloud computing systems. *10th International Conference on Reliability, Maintainability and Safety (ICRMS)* (pp. 435-439). IEEE.

Papazafeiropoulou, A., & Spanaki, K. (2016). Understanding governance, risk and compliance information systems (GRC IS): The experts view. *Information Systems Frontiers, 18*(6), 1251-1263.

Pee, L. G., Pan, S. L., Wang, J., & Wu, J. (2021). Designing for the future in the age of pandemics: a future-ready design research (FRDR) process. *European Journal of Information Systems, 30*(2), 157-175.

Ponce, F., Márquez, G., & Astudillo, H. (2019). Migrating from monolithic architecture to microservices: a rapid review. *2019 38th International Conference of the Chilean Computer Science Society (SCCC)* (pp. 1-7). Concepcion: IEEE.

Poth, A., Schubert, N., & Riel, A. (2020). Sustainability efficiency challenges of modern it architectures–a quality model for serverless energy footprint. *Systems, Software and Services Process Improvement: 27th European Conference, EuroSPI 2020, Proceedings 27* (pp. 289-301). Düsseldorf: Springer International Publishing.

Racicot, L., Cloutier, N., Abt, J., & Petrillo, F. (2019). Quality aspects of serverless architecture: an exploratory study on maintainability. *Proceedings of the 14th International Conference on Software Technologies. Volume 1: ICSOFT*, pp. 60-70. Cham, Switzerland: Springer.

Rafi, S., Akbar, M. A., Sánchez-Gordón, M., & Colomo-Palacios, R. (2022). DevOps practitioners' perceptions of the low-code trend. (pp. 301-306). Helsinki, Finland: ESEM'22.

Rajan, R. A. (2018). Serverless architecture - a revolution in cloud computing. *2018 Tenth International Conference on Advanced Computing (ICoAC)*, (pp. 88-93). Chennai, India.

Rakovic, L., Duc, T. A., & Vukovic, V. (2020). Shadow it and ERP: multiple case study in German and Serbian companies. *JEEMS Journal of East European Management Studies, 25*(4), 730-752.

Reichert, M., & Weber, B. (2012). Flexibility issues in process-aware information systems. In *Enabling flexibility in process-aware information systems* (pp. 43-55). Berlin: Springer.

Roca, J. L. (2019). *Software reliability: a must.* Buenos Aires.

Saher, N., Baharom, F., & Romli, R. (2020). Identification of sustainability characteristics and sub-characteristics as non-functional requirement for requirement change management in agile. *International journal of Scientific & Technology Research, 9*(3), 5727-5733.

Said, I. B., Chaabane, M., Bouaziz, R., & Andonoff, E. (2015). Flexibility of collaborative processes using versions and adaptation patterns. *2015 IEEE 9th International Conference on Research Challenges* (pp. 400-411). Information Science (RCIS).

Salama, M., Bahsoon, R., & Lago, P. (2019). Stability in software engineering: survey of the state-of-the-art and research directions. *IEEE Transactions on Software Engineering, 47*(7), 1468-1510.

Seethamraju, R., & Frost, G. (2016). Information systems for sustainability reporting-a state of practice. *AMCIS 2016 Proceedings*, (p. 1). San Diego.

Sewak, M., & Singh, S. (2018). Winning in the era of serverless computing and function as a service. (pp. 1-5). Pune: 2018 3rd International Conference for Convergence in Technology (12CT).

Shahzad, B., Abdullatif, A. M., Ikram, N., & Mashkoor, A. (2017). Build software or buy: a study on developing large scale software. *IEEE Access, 5*, 24262-24274.

Singh, C., & Pekkola, S. (2021). Packaged enterprise system customization–a systematic literature review. *Proceedings of the 54th Hawaii International Conference on System Sciences*, (pp. 6743-6750).

Slamaa, A. A., El-Ghareeb, H. A., & Saleh, A. A. (2021). A roadmap for migration system-architecture decision by neutrosophic-ANP and benchmark for enterprise resource planning systems. *IEEE Access, 9*, 48583-48604.

Speicher, M. (2015). *What is usability? A characterization based on ISO 9241-11 and ISO/IEC 25010.* arXiv preprint arXiv:1502.06792.

Spurrier, G., & Topi, H. (2017). When is agile appropriate for enterprise software development? *Proceedings of the 25th European Conference on Information Systems (ECIS)* (pp. 2536-2545). Guimarães, Portugal: AIS Electronic Library (AISeL).

Subramanyam, R., Ramasubbu, N., & Krishnan, M. S. (2012). In search of efficient flexibility: effects of software component granularity on development effort, defects, and customization effort. Information. *Systems Research, 23*(3), 787-803.

Taibi, D., Spillner, J., & Wawruch, K. (2020). Serverless computing-where are we now, and where are we heading? *IEEE software, 38*(1), 25-31.

Thambusamy, R., & Salam, A. F. (2010). Corporate ecological responsiveness, environmental ambidexterity and IT-enabled environmental sustainability strategy. *Proceedings of the 31st International Conference on Information Systems (ICIS).* Saint Louis: Association for Information Systems.

Thuan, N. H., Phuong, H. A., George, M., Nkhoma, M., & Antunes, P. (2020). Toward an ontology for improving process flexibility. *International Conference on Future Data and Security Engineering* (pp. 411-428). Cham: Springer.

Tsai, W., Bai, X., & Huang, Y. (2014). Software-as-a-service (SaaS): perspectives and challenges. *Science China Information Sciences, 57*(5), 1-15.

Venters, C., Lau, L., Griffiths, M., Holmes, V., Ward, R., Jay, C., . . . Xu, J. (2014). The blind men and the elephant: towards an empirical evaluation framework for software sustainability. *Journal of Open Research Software, 2*(1).

Verdantix. (2022). *Research portal*. Opgeroepen op January 6, 2023, van Verdantix: https://www.verdantix.com/research-portal

Verdantix. (2023, March 24). *Tech Roadmap: EHS Technologies 2023.* Opgeroepen op March 26, 2023, van https://www.verdantix.com/report/environment-health-safety/tech-roadmap-ehs-technologies-2023

Wang, X., Cao, F., & Ye, K. (2018). Mandatory corporate social responsibility (CSR) reporting and financial reporting quality: Evidence from a quasi-natural experiment. *Journal of Business Ethics,, 152*(1), 253-274.

Wu, C., Buyya, R., & Ramamohanarao, K. (2019). Cloud pricing models: taxonomy, survey, and interdisciplinary challenges. *ACM Computing Surveys (CSUR), 52*(6), 1-36.

Zhang, G., Liu, L., & Guo, H. (2021). Investigating the impact of cloud computing vendor on the adoption of cloud computing. *Mobile Information Systems, 2021*, 1-18.

APPENDICES

Appendix A Data sources

Table 19 Overview of data sources for each component

Name	Offering	Heritage	ESG Type	Pricing model	Deployment model	Architecture	Quality characteristics	Tenancy	Cloud service methods	Coding methods	Founded	ESG Module added
Software 1	Interview	External research platform, vendor website	Interview, vendor website, webinar	Interview, vendor website	Interview	Interview, community platform, email	Interview	Interview, external research platform	Interview, Vendor website	Interview, YouTube	Vendor Website, YouTube	Interview, Vendor website
Software 2	Interview	Interview	Interview, vendor website, webinar	Interview, vendor website	Interview	Interview, vendor website, community platform	Interview	Vendor website, webinar	Interview	Vendor website	Vendor website	Vendor website
Software 3	Interview	Interview	Interview, vendor website, webinar	Interview, vendor website	Vendor website	Vendor website, YouTube	Interview	Vendor website	Vendor website	Vendor website	Vendor website	Vendor website
Software 4	Interview	Interview	Interview, vendor website	Interview, vendor website	Interview	Interview, YouTube	Interview	Vendor website	Vendor website	Interview	Vendor website	Vendor Website, Interview
Software 5	Interview	Interview, vendor website	Interview, vendor website	Interview, vendor website	Interview, vendor website	Interview	Interview	Interview	Interview	Interview	Vendor website	Vendor website
Software 6	Interview	Vendor website	Interview, vendor website	Interview, vendor website	Vendor website	Interview, YouTube	Interview	Vendor website, external research platform	Vendor website, interview	Vendor website	Vendor website	Vendor website
Software 7	Interview	Interview	Interview, vendor website	Interview, vendor website	Interview	Interview	Interview	Vendor website, external research platform	Vendor website	Vendor website	Vendor website	Vendor website

Name	Offering	Heritage	ESG Type	Pricing model	Deployment model	Architecture	Quality characteristics	Tenancy	Cloud service methods	Coding methods	Founded	ESG Module added
Software 8	Interview	Interview	Interview, vendor website	Interview, vendor website	Vendor website, webinar	Interview, webinar, YouTube	Interview	Vendor website, webinar	Interview	Vendor website	Interview	Interview
Software 9	Interview	Interview	Interview, vendor website	Interview, vendor website	Interview, email	Interview, email	Interview	Vendor website, external research platform	Interview	Vendor website	Interview, Vendor website	Interview, Vendor website

Appendix B Interview guide

In this study, we want to learn about future-proof ESG reporting software. The purpose of the study is to learn about the influence of architecture styles on flexibility, robustness, and usability. The study is conducted by Fieke Dhondt, a student in the MSc programme Sustainable Business and Innovation at the Department of Sustainable Development, Utrecht University.

Table 20 Interview guide

Interview section	Question/ topic
Introduction	
Vendor	
	Can you introduce yourself and the ESG reporting software tool you are working for?
	What type of business information systems did *Software x* originally start with? o Business performance management system o Enterprise resource planning system o Governance, risk and compliance system o Environmental, health and safety system o ESG system
	What is the type of pricing model that *Software x* uses? o Subscription base o Consumption base o Pay-as-you go
	How is the solution offered to users? o Off-the-shelf o Modifiable off-the-shelf o Custom developed
	What is the architecture style of *Software x*? o Monolithic o Service-oriented o Microservice o Serverless
	Is the software offered as on-premises or on the cloud?
	Quality sub-characteristics
	Analysability: Is the software tool able to automatically detect a bug or reason for failure?

Interview section	Question/ topic
	Installability: How easy is it to install the software?
	Modularity: How does the software tool support a modular design approach?
	Replaceability: How does the software tool handle the replacement of its components or modules?
	Reusability: Have components or modules from the software been reused by other solutions?
	Interoperability: How is the system integrated with other system?
Implementer	
	Can you introduce yourself and the ESG reporting software tool?
	How many engagements have you worked on with the tool?
	Quality sub-characteristics
	Analysability: Is the software tool able to automatically detect a bug or reason for failure?
	Installability: How easy is it to install the software?
	Modularity: How does the software tool support a modular design approach?
	Replaceability: How does the software tool handle the replacement of its components or modules?
	Reusability: Have components or modules from the software been reused by other solutions?
	Interoperability: How is the system integrated with other system?
Section 1: How does the software system respond to change when new sustainability regulations or business needs were presented.	
	Can you explain how it works if a new sustainability regulation, such as the CSRD, is

Interview section	Question/ topic
	introduced and how the software adapts to include the framework? How would you rank the following statement: When a new sustainability regulation is launched, the system can adjust easily How long does it typically take to implement a new sustainability regulation? * 1- > 8 weeks 2- 7-8 weeks 3- 4-6 weeks 4- 2-3 weeks 5- 1 week Can you explain how it works if a customer demands a new functionality, how does the software tool adapt to include the framework? How would you rank the following statement: When a new user demand is recognised, the system can adjust easily How long does it typically take to implement a new functionality following a user demand? * 1- > 1 year 2- 10-12 months 3- 7-9 months 4- 4-6 months 5- 1-3 months Are new functionalities added for all users or also separately for specific users? 1. all 5. separately
Section 2: Bugs occurring after a change in the system relating to a new sustainability regulation or a user demand.	How would you rank the following statements: How confident are you after changing the tool about the system's - Accuracy

Interview section	Question/ topic
	How confident are you after changing the tool about the system's - Reliability
	Did bugs ever occur after a new regulation framework was added?
	How would you rank the following statement: Unexpected bugs often occur after a change in the system - For a new regulation framework
	Did bugs ever occur when a new functionality for a user was added?
	How would you rank the following statement: Unexpected bugs often occur after a change in the system - To streamline to the user's business process
	Did it ever happen that the software went down after a new functionality was added?
	How would you rank the following statement: The system rarely experiences downtime or outages after a change is implemented
Section 3: Recovering from bugs.	
	Relating to recovering from bugs, how would you rank the following statement:
	The system can recover from failure or outages - Effective
	On estimate, how long does it typically take to resolve system issues or bugs following a change in the system? 1- > a day 2- 9-23 hours 3- 6-9 hours 4- 3-6 hours 5- 1-3 hours
Section 4: Usability of the system – only for implementers	
	Learning to operate the system went quickly
	Learning to operate the system is easy

Interview section	Question/ topic
	The system is easy to use
	The system is unnecessarily complex
	The system is easy to navigate through
	The system is easy to access
	The system is easy to implement in a client's organisation

Note: From section 1 onwards all scores range from 1 strongly disagree to 5 strongly agree unless indicate differently

Appendix C Coding tree

Table 21 Coding tree

Codes			
Architecture			
	Cloud provider		
		PaaS	
			Apps
			Platform of platforms
		SaaS	
	Data architecture		
		Data columns	
	Microservice		
		Cloud	
		On-prem	
	Monolithic		
	Overarching		
		Cloud	
		On-prem	
	Serverless		
	Service-oriented		
		Cloud	
		On-prem	
	Tenancy		
		Single tenant	
Competition			
Flexibility			
	Adjusting functionality		
		Instant adjustable	
		Paid add-ons	
		Software enhancement	
	Adjusting reporting		
		Calculations	
		Configured	
		Drag and drop	
		Partnership	
		Set up base	
		Template based	
	Business value		
		Visibility	

Codes		
	Configurable	
		Dev Ops
		In-house admin
		System implementer
		Workflow
	Customisation	
		Complexity
		Custom codes
		Proper design
		Setting boundaries
	Ledger	
	Low code, no code	
		no code configuration
	Scalable	
Need		
	Auditable workflow	
	Business value	
	Collecting data	
	Linking frameworks	
	Measure data	
Price		
	Pricing type	
		Consumption based
		Pay-as-you-go
		Subscription base
Quality characteristics		
	Analysability	
	Installability	
		Web-based
	Modularity	
	Replaceability	
	Reusability	
	Interoperability	
		API
		Not connected
Regulations		
	CSRD	
	Custom frameworks	
		Agnostic approach
	Evolving landscape	

Codes				
	GRI			
	Metric Management			
	Out of the box			
	PCAF			
	SASB			
	TCFD			
Robustness				
	Bugs			
		Business processes		
		Configuration issue		
		Custom code		
			Complexity	
			Highly customised	
		Data bug		
		Error messages		
		Regulations		
		Software issue		
		Testing		
	Outages			
		Architecture related		
		Downtime		
	Quality Control			
		Accuracy		
		Reliability		
	Recovering from bugs			
		Cloned environments		
		Refactoring codes		
			Updating calculations	
		Troubleshooting		
			Service	
				SLA
	Releases			
		Compatibility with customisation		
	Security			
	User modification			
		Testing		
			Test environment	
				Feature flag
				Scope environment

Codes		
Software type		
	BPM	
	Carbon accounting	
	EHS	
	ESG add-on	
		Acquired software
		Merged
	GRC	
	LCA	
	Overarching platform	
	Sustainability & Compliance	
System offering		
	Custom development	
		New application
	Modifiable off the shelf	
	Off the shelf	
		SKU
	Out-of-the-box solutions	
		In-house capabilities
Usability		
	Accessibility	
	Adoption	
		Cultural change
	API	
		External provider
	Assistance	
	Complexity	
		Client requests
	Deployment	
	Implementation	
		Implementation model
	Intuitive	
		Customising
		Easy to use
		Navigating through
	Learning	
		Training
	Mobile app	
	Offered languages	

Codes	
	Onboarding
	Used as intended
	UX UI

Appendix D Overview of results

Table 22 Overview of results

Name	Heritage	ESG Type	Offering	Pricing model	Deployment model	Architecture	Quality characteristics	Tenancy	Cloud service methods	Coding methods	Founded	ESG Module added	Flexibility	Robustness	Future proof	Usability
Software 1	GRC	ESG Reporting Software	Modifiable off-the-shelf & Custom developed	Subscription	Cloud, On-premises	SOA, MSA	All	Multi-tenant & Single-tenant	PaaS, SaaS	Low code, no code	2004	2021	High	High	High	High
Software 2	ESG	ESG Reporting Software	Off-the-shelf	Consumption based	Cloud	MSA, Serverless	All	Multi-tenant	SaaS		2004	2004	High	High	High	High
Software 3	EHS	ESG Reporting Software	Modifiable off-the-shelf	Subscription	Cloud	MSA, Serverless	All	Multi-tenant	SaaS		2000	2017	High	High	High	Medium
Software 4	BPM	ESG Reporting Software	Modifiable off-the-shelf	Pay-as-you-go	Cloud, On-premises	SOA	All	Multi-tenant	SaaS	Low code, no code	2005	2022	High	High	High	Medium
Software 5	GRC	ESG Reporting Software	Modifiable off-the-shelf	Subscription	Cloud, On-premises	SOA	All	Single-tenant	PaaS	Low code, no code	2008	2021	High	High	High	Medium
Software 6	GRC	ESG Reporting Software	Modifiable off-the-shelf	Subscription	Cloud, On-premises	Monolithic, SOA, MSA & Serverless	All	Single-tenant	SaaS	Low code, no code	1999	2021	High	Medium	Medium	Medium
Software 7	GRC	ESG Reporting Software	Modifiable off-the-shelf & Custom developed	Subscription & Consumption-based	Cloud, On-premises	SOA	NA	Multi-tenant & Single-tenant	SaaS		2004	2022	High	Medium	Medium	Medium

Software 8	ESG	Carbon Management Software	Off-the-shelf	Subscription	Cloud	MSA, Serverless	All	Multi-tenant	SaaS	2020	2020	Medium	High	Medium	Medium
Software 9	ESG	Product Lifecycle Assessment Software	Off-the-shelf	Subscription	Cloud	MSA	All	Multi-tenant & Single-tenant	*Becoming SaaS*	2001	2001	Medium	High	Medium	NA

Note: MSA = Microservice architecture, SOA = Service-oriented Architecture