Organizational use of low-code development platforms

- The power of the Power Platform

MSc in business administration and e-business

Master thesis

17/05/2021

Marcus Bjørn Larsen – 110702

Supervisor: Jacob Nørbjerg

Characters: 175.971

Pages: 80

Abstract

Purpose - The purpose of this paper is to explain how the use of low-code development platforms to build business applications is impacting organizational interests

Design/Methodology - The paper takes an interpretivist stance basing the research on a case study of the professional services firm – EY and their use of the Microsoft Power Platform.

Findings – The study shows that three use case scenarios exist in the organization – "Professionals", "Taskforce" and "Cowboys" and that each of these appraoches have significant affects to the overall goals of the organization. The study further finds that the implementation of the Power Platform in the organization changes the structure of the IT organization to a whole new archetype of bimodal IT – Decentralized agile IT.

Research limitations - The study is based on the use of a single low-code development platform within one single organization. If a larger representativeness is sought, it will involve more respondents.

Practical implications - The paper helps organizations in understanding the impact of introducing low-code development platforms widely in the enterprise.

Originality - This paper adds to the literatures of Low-code development, IT governance, bimodal IT as well as software engineering, by providing empirical evidence of low-code use case scenarios along with their impact on organizational interests.

Contents

Introduction

Low-code development is a concept that is rapidly taking over the industry of application development and hyper automation (Rashid, 2021). Low-code development is in many ways the reinvention of Rapid Application Development, which was introduced in the 80s as alternatives to traditional programming techniques. Essentially RAD tools were focusing on constrained functionality, user experience and performance, which resemble many of the ideas behind low-code today (Rashid, 2021).

According to Gartner 65% of all app development will be done through the use of low-code application platforms by 2024 and companies are increasingly being expected to address low-code development on a strategic level (Goswami, 2021). Companies like Microsoft, Salesforce and Outsystems are acknowledged as some of the lead providers of low-code development platforms in both Gartner's Magic Quadrant and the Forrester Wave (Vincent, 2019; Rymer & Koplowitz, 2019). Outsystems define a low-code platform as: "(…) a collection of tools that enable the visual development of applications through modeling and a graphical interface" (Alexander, n.d.). These platforms are increasingly being adopted by companies in developing enterprise applications and automating workflows. Especially the graphical interface allows for these applications and workflows to be developed by people with close to no experience within application development – also known as "Citizen Development" (Oltrogge et al., 2018).

Gartner defines citizen development as: "(…) a user who creates new business applications for consumption by others using development and runtime environments sanctioned by corporate IT" (Gartner, n.d.). The citizen developer represents the move from developing standard Excel spreadsheets and Access databases for local use into developing applications for the wider organization (Gartner, n.d.). To capture the value from low-code development platforms organizations need to rethink their strategies, leadership and ways of working as enterprise-wide adoption of such platforms does not come by itself (Goswami, 2021).

These platforms enable the entire organization to build business applications more rapidly, however challenges in controlling development efforts arise with the decentralization of business application development. Although the discussion around centralization versus decentralization are widely covered in IT governance literature, little focus has been given to the impact of low-code development in organizations (Weil & Ross, 2004).

The lacking understanding of application development by citizen developers infer risks on the business as there is no guarantees that these will follow software engineering best practices. The academic area of software engineering has been developed through many years of research and the decentralization of this activity seem to neglect this (Humphrey, 1975; Sommerville, 2015). The

concept of Low-code development has risen out of the business, which has led to a general lack of research on the academic area of low-code development and the impact it has on organizations. Research in the area can help bring issues to light which the businesses using these platforms might not be aware of.

Thus, this paper aims to answer the question:

> How does the use of low-code development platforms to build business applications impact organizational interests?

In order to answer this question two sub questions, need answering. First of all, the ways in which these low-code applications are being used needs to be uncovered providing insight to whether best practices of software engineering are followed and the impacts this has. Second the governance mechanisms and structures need to by considered in assessing how organizations cope with these use cases. This lead to the following sub-questions:

How are low-code development platforms governed in organizations?

How are employee developers using low-code development platforms?

This paper will first review existing literature in the areas of low-code, business interests, IT governance and software engineering. Then a walkthrough of the methodology and methods followed in the research process, along with a description of the case organization. Following this an analysis of three use case scenarios identified through this paper will be made mapping out the purpose and process of developing business applications in each of the three scenarios. Then an analysis will be conducted answering the first sub-question by investigating how the use cases in the case organization are following best practices of software engineering. In answering the second sub question the governance mechanisms and structures will be analyzed providing insight to how the case organization cope with this platform. The outcomes of the two analysis are brought together in a discussion of the findings and limitations of this paper. The paper is ended with a conclusion on the original research question taking limitations of the paper into consideration.

Literature review

This section will review the literature related to the topic of the research. First the concept of low-code development will be reviewed focusing on explaining what constitutes low-code, what the benefits and consequences are and how it is implemented in businesses. Following this the best practices of software engineering will be outlined considering three approaches – agile, plan-driven and service-oriented engineering. Then the concept of business goals will be reviewed in terms of how they are developed in organizations along with some typical organizational goals. Finally, IT governance literature will be reviewed on the concepts of governance mechanisms and bimodal IT.

Low-code development

The term Low-code development was first coined by Forrester in 2014 and used to describe platforms which with minimum amount of code enabled the user in building software solutions (Forrester, 2014). The concept underlying the term resembles other well-known concepts in terms of Rapid Application Development (RAD) and fourth generation programming languages (4GL) (Virta, 2018). The underlying focus of these concepts are to deliver software faster through higher abstracted programming languages, which constitutes the motives behind low-code development (Martins et al., 2020).

Benefits of low-code development

Martins et al. (2020) demonstrate the benefits of low-code development through a case study, where a group of developers build an HR self-service portal using "Outsystems". Their results show that main benefits include acceleration of development, reduction of IT backlog and responsiveness and lower dependance on tech skills (Martins et al., 2020). Waszkowski (2019) paint a similar picture using the low-code platform Aurea BPM for automating business processes in manufacturing. Woo (2020) illustrated how New York City officials built an online portal for collecting information from COVID-19 patients in just three days. This was done without a single line of code using a low-code development platform called Unqork (Woo, 2020). The speed of delivery and lower dependency of low-code is further supported by Arora et al. (2020), who evaluates a low-code platform called "Sagitech studio" emphasizing the features of the platform which reduce time spent on software projects and increase speed of delivery (Arora et al., 2020). Zolotas et al. (2018) describe a low-code platform called "RESTSec", which in line with the others emphasize the ease of use as well as the rapid delivery of working software (Zolotas et al., 2018). All seem to agree that low-code development platforms shorten the development-cycle and reduce dependency on IT skills.

Challenges of low-code development

On the other hand, Oltrogge et al. (2018) assess security impacts of mobile apps developed by citizen developers using low-code app generators. They investigate this by first reverse engineering free apps on google play and identify which are developed using such a generator and second investigate the autogenerated code of each of the identified platforms. They found that around 11% of the apps that are available for free are developed using such a platform and that the bulk of these platforms had major security risks such as oversharing of rights and vulnerability for http attacks (Oltrogge et al., 2018). A very valid point they make is also that errors and security risks are amplified due to the auto generated code being used in multiple apps, on the other hand if the platform developers are making an effort to secure the autogenerated code this effect is also amplified.

Khorram et al. (2020) also introduce three categories of challenges in their investigation of using low-code testing tools. The first one is the role of citizen developer and her low-level technical knowledge in the testing activities, where they highlight the importance of the inclusion of the business early on, as their requirements define the testcases and consequently the limitations of their non-technical background. The second category is the importance and consequently the challenges in offering high-level test automation. They highlight that automatic testing exist in form of "Data-Driven", "Model-Based", and "Record and Replay", but most of them still require some manual scripting, which does not align with the low-code development techniques. The final category is leveraging the cloud for executing tests alongside supporting testing of cloud-based applications, where they highlight that there is a need for testing cloud-based applications and not many tools exist for automatic testing in the cloud (Khorram et al., 2020).

Sahay et al. (2020) evaluated eight low-code development platforms and found four key challenges transcending these. The first one is Low-code platforms' interoperability, where they highlight the risk of vendor lock-in as solution architectures are not easily transferred between the platforms. The second is extensibility, where they highlight that most platforms do not allow for the addition of new functionality. Learning curve is the third one, which concerns the less intuitive user interface of these platforms inhibiting adoption. The final challenge is scalability, which is challenging due to lacking open standards with the providers making assessment of scalability dificult (Sahay et al., 2020).

Low-code development in businesses

Ploder et al. (2019) investigates whether knowledge workers are accepting the trend of low-code/no-code and use the tools provided. They suggest that clear communication is needed about the benefits of such platforms in fostering adoption within the organizations. However, they also find that

knowledge workers are willing to learn these new tools and are ready to embrace it (Ploder et al., 2019).

Sahay et al. (2020) provide a set of features for businesses to evaluate low-code development platforms when selecting a provider for the specific problem at hand. They identify 35 features pertaining to 10 categories: graphical user interface, interoperability support, security support, collaborative development support, reusability support, scalability, business logic specification mechanisms, application build mechanisms, deployment support and kinds of supported applications (Sahay et al., 2020). Gartner (2020) assess vendors across 15 evaluation criteria pertaining to two dimensions: ability to execute and completeness of vision. Based on these dimensions Gartner maps low-code development platforms in their magic quadrant where Outsystems, Microsoft, Mendix and Salesforce are clear leaders. Forrester (2019) also maps low-code development platforms in their equivalent to the magic quadrant of Gartner. They evaluate low-code development platforms against 28 criteria grouped in three categories: Current offering, Strategy and market presence. Their assessment paint a similar picture putting the same four low-code development platforms as the leading providers in the market (Forrester, 2019).

Even though some literature has been written on low-code development including specific use cases, challenges and opportunities as well as considerations when selecting and implementing a platform. Little research exist on how these platform in fact are being used in the business. One source investigating this is a thesis by Virta (2018) who investigates how low-code development relates to standard software development through a case with a Finish consultancy - Biit oy - and their use of Salesforce. She discovers that low-code tools are perceived to reduce development cycles and offer simple process automation, however they also percieve them to be inefficient, have performance issues with large data volumes, provide maintenance issues with complex sollutions and have an obscure nature to their offered tools.

In sum, available literature in the area of low-code development suggest clear benefits of reduced development cycle and dependency on IT skills. Challenges of using low-code development platforms pertain to security, testing and the platforms themselves. In business contexts low-code development is still in its infancy and clear communication is needed to foster adoption.

Software engineering

The Software engineering process was defined by Humphrey (1989) as: “(…) the total set of software engineering activities needed to transform a user’s requirements into software". Steps in this process

includes as appropriate “(…) requirements specification, design, implementation, verification, installation, operational support, and documentation" (Humphrey, 1989). Humphrey also introduce the concept of "Software Process Architecture", which is a type of metamodel in how to conduct software engineering. These metamodels can take three forms, waterfall, incremental and integration and configuration (Sommerville, 2015).

The plan-driven approach assumes software engineering to be a linear process from planning to implementation, which is mostly relevant for a) Embedded systems where the software has to interface with hardware systems, b) Critical systems where there is a need for extensive safety and security analysis of the software specification and design or c) Large software systems that are part of broader engineering systems developed by several partner companies (Sommerville, 2015).

The incremental approach assumes software to be broken down into smaller increments that are then concurrently worked on as smaller pieces of software. Three major advantages come from following this approach: a) The cost of implementing requirements changes is reduced, b) It is easier to get customer feedback on the development work that has been done and c) Early delivery and deployment of useful software to the customer is possible, even if all the functionality has not been included (Sommerville, 2015). Three common issues arise following agile methods: Lacking documentation, keeping customers involved and development team continuity.

The integration and configuration approach assumes the elaborate reuse of existing software to be a key driver of the engineering process. Although reuse can happen across all approaches, this approach takes a more structured and formal approach to reuse of software. Three types of software components are frequently reused a) Stand-alone application systems that are configured for use in a particular environment, b) Collections of objects that are developed as a component or as a package to be integrated with a component framework such as the Java Spring framework and c) Web services that are developed according to service standards and that are available for remote invocation over the Internet (Sommerville, 2015).

Regardless of software engineering approach, a standard set of activities must be conducted. These activities include software specification, software development, software validation and software evolution (Sommerville, 2015).

Software specification

Software specification entails specifying what the software should be doing. As a part of this process, the requirements of the system should be defined. Requirements can be categorized as user

requirements, which entails more high-level requirements and system requirements, which are directly related to the system and its functionality. These requirements can be further categorized as either functional or non-functional requirements. Functional requirements are statements of the services that the system should provide, whereas non-functional requirements act as constraints on the services or functions offered by the system. Non-functional requirements can be further divided into product requirements (specifying constraints on the runtime behavior of the software), organizational requirements (broad system requirements derived from company policies or processes) or external requirements (requirements derived from external factors).

Software specification starts with requirements elicitation and analysis, which usually follows four process steps: requirements discovery and understanding, requirements classification and organization, requirements prioritization and negotiation and requirements documentation. Techniques in doing this include interviewing stakeholders, ethnography and observation of stakeholders and defining stories or scenarios to capture the functionality of the system.

The requirements specification is the second step in the software specification and entails noting down the requirements discovered in the previous step. This can be challenging due to the various interpretations different stakeholders can have. These requirements could be described using natural language, structured specifications and use cases as well as graphical and mathematical notations. Graphical notations could include context models (context diagrams and business process models), interaction models (use case diagrams and sequence diagrams), structural models (class diagrams, generalized class diagrams and aggregated class diagrams) or finally behavioral models (data-driven modelling and event-driven modelling). Diagrams such as these are used to facilitate discussions about the requirements and help stakeholders understand what the system should do. In the end this step should end with a software requirements specification document.

The final step is requirements validation which entails checking that the noted requirements, define the system that the customer wants. This includes conducting validity checks, consistency checks, completeness checks, realism checks or verifiability. Several techniques exist to aid in conducting this step such as prototyping, requirements review or test case generation.

During the entire process, changes to requirements needs to be tracked and managed. This is done by agreeing on requirements identification, a change management process, traceability policies and tool support to aid in requirements storage, change and traceability management. Whenever a change is suggested then a problem analysis and change specification is conducted, followed by a change analysis and costing discussions, and finally change implementation (Sommerville, 2015).

In terms of software specification, requirements are hardly specified when taking an agile approach as focus is put on conducting the work. Requirements are instead captured through "user stories", which are scenarios of use that are defined in a collaborative manner between the system customer and the development team. These are then put in "story cards" and the development team further breaks the story down into tasks along with an estimation of the effort needed to complete that

task. These are usually validated with the customer ensuring the right understanding of the user story (Sommerville, 2015).

In service-oriented engineering specification is done by mapping existing processes along with the data that flows through and then designing an ideal workflow. Based on the ideal workflow, a search process is initiated where various services with the capabilities to solve parts of the process is identified. The main goal of this approach is reuse of existing software and thus it is generally accepted that the "ideal" requirements must be modified to fit the services that are available (Sommerville, 2015).

Software development

Another process in software engineering is the **software development**. This entails architectural design, database design, interface design and component selection and design and finally the actual programming of the system. The software specification serves as a foundation for this process, however in the case that the environment have not been mapped out in context diagrams and use case diagrams through the previous step, this would need to be done first. This helps the designers of the system to understand what features are contained in other systems as well as what the system is intended to do.

Architectural design is based on the requirements specified through the specification process and they serve as a guide in how the system should be structured. Non-functional requirements often influence key architectural decisions prioritizing what quality attributes such as performance, dependability, security etc. should be prioritized. The architecture should be documented from different architectural views such as Krutchens 4+1 entailing a logical, process, development and physical view or Hoffmeister adding a conceptual view to it (Sommerville, 2015). This also includes decisions on what architectural patterns should be applied. An Architectural pattern is an abstraction of best practice constituting different designs that have been tested in different systems and environments. Each pattern is fit for a specific situation such as the model-view-controller pattern is good for when you have multiple views and ways of interacting with the same data or pipes and filters when you are processing data in sequences based on user actions. Another consideration in this process includes whether the system in question follows a general application architecture, such as information systems almost always include an interface, an authorization layer, information retrieval and modification and a transaction management database. Application architectures can be used as 1) a starting point for the architectural design process, 2) a design checklist, 3) a way of organizing the work of the development team, 4) a means of assessing components for reuse and 5) a vocabulary for talking about applications.

Database design includes identification of the objects in the system along with their properties. Ways of identifying these include grammatical analysis of a natural language description of the system, use tangible things in the applications domain and scenario-based analysis.

Interface design entails specifying the signatures and the semantics of the services that are provided by the object or by a group of objects. The representation of the data is kept out of this part as it eases the process of changing the representation, but the actions to interact with the data should be included.

Implementation entails the actual development of the system. Some things that are not often considered in this process is the reuse of existing software, configuration management and host-target development. Reuse can occur on four levels: abstraction level (reuse of knowledge of successful abstractions), object level (reuse of objects from a library), component level (reuse of collections of objects and object classes that operate together to provide related functions and services) and system level (reuse of entire application systems). Configuration management entail four fundamental activities: version management (keep track of the different versions of software components), system integration (define what versions of components are used to create each version of a system), problem tracking (report bugs and track who fixes them) and release management (planning the functionality of new releases and organizing the software for distribution). Host-target development refers to a development platform (host) and an execution platform (target). How to deploy the software on the target platform needs to be considered through three key issues: the hardware and software requirements of a component (are these requirements fulfilled on the target platform), the availability requirements of the system (should back-up systems be deployed on other platforms) and component communications (are there any communication latency issues) (Sommerville, 2015).

In terms of software development (design and implementation), not much effort is put in to elaborate modelling and design of the system following agile methodologies. The user stories serve as software increments that will be developed throughout an iteration. The customer will prioritize the user stories, and the development team will develop the system increments based on that prioritization (Sommerville, 2015).

In service-oriented engineering very little effort is put into the development phase as the entire workflow has been mapped and services has been identified in the specification phase. Based on those the services are connected and configured for the purpose of the application (Sommerville, 2015).

Software validation

The third element that all software engineering models should include is the **software validation & verification.** Software validation checks for are we building the right things and software verification checks for are we building things right. The purpose is to establish confidence that the software is fit

for purpose. How much a confidence is needed depends on the criticality of the software, the user expectations as well as market considerations on competing software. This process can involve software inspections, which are inspections of the static elements of the system such as code and system models. It also entails software testing which is testing the system executables in runtime. Testing can take three forms: development testing, release testing and user testing.

Development testing is conducted during the development process and can be either unit testing (focus on testing the functionality of objects or methods), component testing (focus on testing the component interfaces that provide access to the component functions) and system testing (focus on testing component interactions).

Release testing is conducted in the end of the development process and prior to the release of the system. The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use. Three approaches can be taken to this type of testing: requirement-based testing (testing the functionality fulfills requirements), scenario testing (create typical use cases and go through them in the system) and performance testing (testing whether the system is performing under high pressure).

User testing is sometimes performed during the development process and sometimes after. This type of testing is concerned with the users' experience of the system and allow them to provide feedback. Three types of user testing are alpha testing (selected group of users work closely with the development team in testing early releases), beta testing (a larger group of users are allowed access to the system for experimenting) and acceptance testing (customers are testing the system and decides whether it is ready for deployment in their own environment) (Sommerville, 2015).

Testing is also done differently in agile methodologies as no formal software specification exists, it is hard to develop testcases and follow formal test practices. One approach is to follow the test-first development principle or test-driven development. Following this type of testing, test cases are defined and created as executable unit tests by the development team prior to the development of the unit. User testing is then done on the user story following the approach of acceptance testing. Another approach is using pair programming where two developers create code on the same screen, minimizing errors. No official approach is defined for agile software engineering and as such a mix of plan-driven and agile methodologies are often applied (Sommerville, 2015).

There is no specific approach of testing in Service-Oriented engineering either, leading to the same outcome as with agile testing. However, several challenges arise in this stage of service-oriented software engineering. As the source code is not visible to the user, regular testing based on the source code is not an option and furthermore debugging is harder as the failure could come from the service provider. The services are under the control of an external vendor meaning that the developer has no control of the service, using it often comes with a price and performance of the service is not guaranteed as multiple users can overload the vendor’s system (Sommerville, 2015).

Software evolution

The fourth and final element in any software engineering process is **software evolution.** This process is concerned with the system's further development and maintenance after release.

The evolution process includes taking in change requests (formal and informal), assessing their impact on existing software as well as business benefits, implementing the change and update specification materials. Sometimes the change is urgent, and the implementation takes precedence over a comprehensive analysis and update of specification.

Maintenance of a system includes fault repairs (to fix bugs and vulnerabilities), environmental adaptation (to adapt the software to new platforms and environments) and functionality addition (to add new features and to support new requirements).

If proper evolution processes are not followed, the developed systems risk becoming out of date and in the end legacy systems. Legacy systems can be very costly for companies to maintain, but they can also be very costly to convert into modern technology (Sommerville, 2015).

Following agile methodology, new user stories are created following the same process as before the system was released. As the system is developed concurrently in small increments, it is easy to adapt to change. However, the structure of the system risk becoming less coherent, and code can be duplicated across the system. One way of preventing this suggested by extreme programming methods is for developers to constantly refactor the code, when opportunities for improvement are discovered (Sommerville, 2015).

Further evolution of a solution developed using service-oriented engineering is not specified by Sommerville (2015). As such methods described in agile and plan-driven approaches can be applied filling this gap. One exception is that developers do not have the same control over the individual services and leading to solutions being dependent on the vendors' evolution processes (Sommerville, 2015).

Business interests

The concept of business interests or organizational goals is a heavily complex topic. In traditional economic theory organizations are seen as one single entity working towards the same goal - Profit maximization (Kotler et al. 2018). Cyert and March (1963) argued that organizations are not one single entity but a collective of individuals with each their own goals. According to them organizational goals are:

> "(…) the result of continued bargaining and stabilization processes among relevant parties within the firm, leading to multiple dimensions along which organizational goals are set, including production, inventory, sales, market share and profitability" (Kotler et al. 2018:4).

This is further supported by Gross (1969) who in his study of 80 universities found that different employees had different views on what the goal of that university was (Gross, 1969). The interests of

an organization is thus a combination of various different local interests within the organization. This is further supported by Eliashberg & Michie (1984) as they quote Lilien & Kotler (1983) "An organization is a complex hierarchical social system pursuing a variety of organizational and personal goals" (Eliashberg & Michie, 1984).

Kotlar et al. 2018 critically reviewed the literature and conceptualized a model to illustrate the nature of organizational goals:

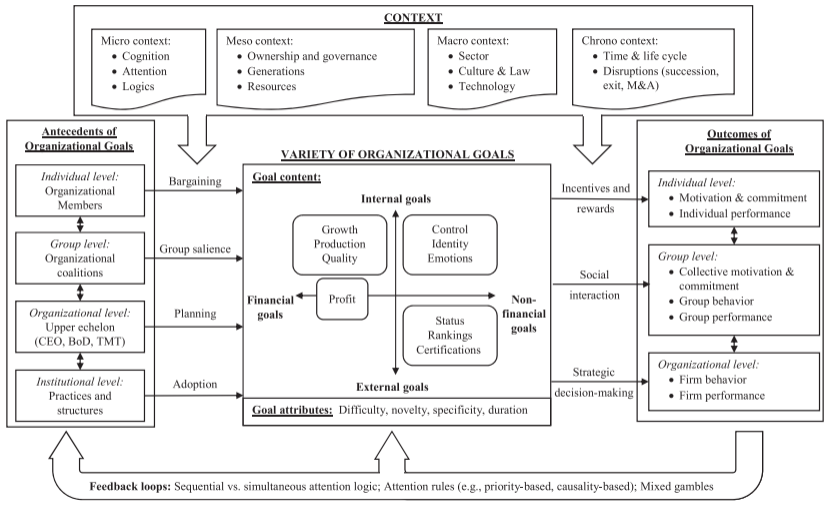

Figure 1. Organizational goals: variety, antecedents, outcomes and context

As their model shows, the variety of organizational goals can be organized along two dimensions: the nature of the goal (financial or non-financial) and the source of the goal (internal or external). The variety of goals are constructed by the antecedents of organizational goals on four levels: individual, group, organizational and institutional level. The construction of this variety of goals lead to outcomes on three different levels: individual, group and organizational. The processes connecting the antecendants and outcomes to the variety of organizational goals are influenced by the context of four different areas: micro, meso, macro and chrono. The outcomes, antecendants and variety of organizational goals are furthermore all connected in a feedback loop underlining the dynamicity of organizational goals (Kotlar et al. 2018).

They list 10 organizational goals across their own dimensions. Internal financial goals include Growth, production, quality and in part also profit as this can be seen as both an internal and external goal. Internal non-financial goals include control, identity and emotion and finally external non-financial goals include status, rankings, and certifications. Eliashberg & Michie (1984) refer to Kotler (1980) who also found profit and growth as common organizational goals, but further adds market share, risk diversification, and innovation to the list (Eliashberg & Michie, 1984). Stagner (1969) further expands the list by including organizational cohesiveness to be an important goal for executives and organizational performance (Stagner, 1969).

Managing organizational goals

The complex and dynamic nature of organizational goals justifies strategic management as an area of research. In setting the long-term goals of an organization literature suggest the use of vision, which clearly defines the management's aspirations for the business. This should be accompanied by a mission describing why the organization are doing what they are doing and values, which defines how the organization is conducting business (Thompson et al., 2016). These are often followed by a strategy, which constitutes an action plan for the immediate future (Thompson et al., 2016). The definition and communication of these elements in a business are meant to align the business across all levels in achieving the same organizational goals. Figenbaum et al. (1996) conceptualized a three-dimensional matrix of organizational goals intended to help managers creating a "strategic reference point" for organizational goals. These dimensions concerned the internal capabilities considering strategic inputs and outputs, the external conditions considering competitors, customers, and stakeholders and finally, time considering the past and future (Figenbaum et al. 1996). Organizational goals can thus be related to building internal capability, accommodating, or adapting to external conditions and learning from the past or preparing for the future. Kaplan & Norton (2007) also provided a way of organizing both financial as well as long term strategic goals across different levels of the organization through their balanced scorecard approach.

In sum organizational goals are set through dynamic processes happening at multiple levels and across multiple departments of an organization. In aligning these goals strategic managements literature suggests the definition and communication of vision, mission, values, and strategy as these help employees understand the desired direction of the organization. The definition of these elements thus suggests the overall goals of the organization.

IT governance

IT governance is a concept focusing on how top managers ensure that IT related efforts in the organization are coordinated in coherence with the overall business strategy (De Haes & Van Grembergen, 2004). De Haes & Van Grembergen (2004) define IT governance as:

> "(…) an integral part of corporate governance and addresses the definition and implementation of processes, structures and relational mechanisms in the organization that enable both business and IT people to execute their responsibilities in support of business/IT alignment and the creation of business value from IT-enabled business investments" (De Haes & Van Grembergen, 2004:1).

IT governance is thus the responsibility of corporate management and aims to achieve business and IT alignment.

Defining IT Governance in organizations

Weil and Ross (2004) provide a framework to define the IT governance of an organization in one single page. This framework is a five by six matrix consisting of five key issues that any firm need to consider in defining IT governance and six decision making archetypes ranging from centralized to decentralized. The five key issues are 1) IT principles, 2) IT Architecture, 3) IT Infrastructure, 4) Business application needs and 5) IT investment and prioritization. The six archetypes are 1) Business monarchy, 2) IT Monarchy, 3) Federal, 4) IT duopoly, 5) Feudal and 6) Anarchy. The map of how decisions are made in the organization should be followed with a coordinated set of governance mechanisms. These mechanisms usually come in three types: 1) Decision making structures, 2) alignment processes and 3) formal communications (Weil & Ross, 2004).

The three types of mechanisms defined by Weil & Ross (2004) are closely related to the structures, processes, and relational mechanisms in an organization that De Haes and Grembergen (2004) suggest. Structures focus on defining clear roles and responsibilities, how the IT function is organizationally structured and setting down IT strategy and steering committees. Processes focus on aligning business and IT. They suggest several tools to achieve such an alignment, including Strategic Alignment model, balanced scorecard, Information Economics, Service Level Agreements, COBIT and ITIL as well as different maturity models. Relational mechanisms focus on building a good collaboration between IT and business by fostering knowledge sharing, career crossovers, cross-training etc. (De Haes and Grembergen, 2004).

IT Governance structures in top performing companies

In analyzing top performers Weil & Ross (2004) find that governance types are closely related to performance goals. Companies aiming to increase growth define a decentralized governance structure fostering local innovation and customized capability. In contrast, companies aiming to increase profit define a centralized governance structure fostering reduced business cost and increased shared infrastructure. Companies aiming at increasing asset utilization define a hybrid of the two governance structures fostering lower IT unit costs and to some extent shared IT services (Weil & Ross, 2004).

Balancing the paradoxes

In the pursuit of being both efficiency and flexibility organizations often adopt the structure of bimodal IT organizations. Bimodal IT entails having two modes of the IT department – traditional and agile IT. Traditional IT is seeking to achieve efficiency in the use of technology within the

organization and agile IT is seeking to be innovative in the use of technology (Jöhnk et al., 2019). The implementation of Bimodal IT often results in transformational challenges stemming from the disruption of existing structures in the IT organization. Furthermore, operational challenges arise from having a bimodal IT organization stemming from the two modes seeking to achieve different goals (Jöhnk et al., 2019).

The nature of bimodal IT is a paradox as high IT spend in achieving innovation contradicts reducing IT cost to achieve efficiency. Jöhnk et al. (2019) adopts the governance mechanisms as defined by De Haes & Van Grembergen (2004) in analyzing the inner workings of two organizations' bimodal IT departments. They found five categories of transformational challenges and six categories of operational challenges resulting from the implementation of bimodal IT. By analyzing the governance mechanisms through the framework of De Haes & Van Grembergen (2004), they find the governance mechanisms to be either Impeding or coping with the transformational and operational challenges (Jöhnk et al, 2019).

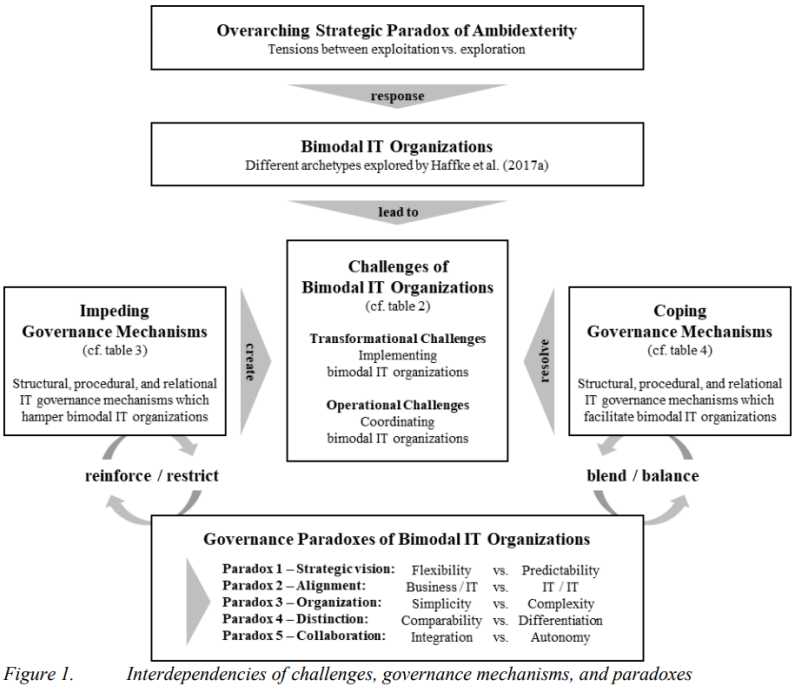

Figure 1. Interdependencies of challenges, governance mechanisms, and paradoxes

The contradictory nature of impeding and coping mechanisms brought five additional paradoxes to light related to strategic vision, alignment, organization, distinction, and collaboration (Jöhnk et al., 2019). The impact of the two categories of governance mechanisms on the paradoxes was found to be either restricting, reinforcing, blending or balancing the paradoxes in the Bimodal IT organizations.

Summing up on IT governance, the primary role of IT governance is to ensure that IT efforts are aligned with business interests. Defining a proper IT governance entails introducing proper structures, processes and relational mechanisms fostering business IT alignment. The introduction of Bimodal IT

introduces some challenges and paradoxes that needs to be managed through proper IT governance mechanisms.

Methodology

Philosophy of science

This paper follows the epistemology of interpretivism as it seeks to understand the phenomenon of how low-code development platforms are impacting the business interests of an organization. In researching this, this paper investigates how these platforms are being used by different employees in the organization, while simultaneously investigating the governance mechanisms of said platforms. In taking this approach the author is aiming to "(…) understand the fundamental meanings attached to organizational life" (Saunders et al., 2007:113). The author could have taken a positivist stance in investigating the topic, which would have resulted in a different approach to the research. In this case the research would be focusing on quantifying the impacts on organizations' interests, which would restrict organizational interests to quantifiable measures such as revenue or profit. On the other end of the scale a constructivist stance could have been taken, which would have led the research to be focused on discussions about how individuals in the organization are interacting to construct a sense of meaning and value in using low-code development platforms. The reason that the author takes an interpretivist stance is because the fundamental interest is to understand how the employees' use of low-code development platforms aligns with the overall interests of the organization.

By taking an interpretivist stance this research limits itself to understand the phenomenon through the eyes of the individuals working with or governing the platforms. The researcher thus has to make an effort to "(…) enter the social world of our research subjects and understand their world from their point of view" (Saunders et al, 2016:107). The interpretivist understands that every organization, person, and situation is unique and thus generalizability of the study is compromised. However this quality criteria is of less importance in interpretivism as organizations are unique and the world is ever-changing which means that what is true today might not be in three months - regardless of research approach (Saunders et al., 2007).

This paper follows the interpretivist intellectual tradition of symbolic interactionism which in its essence mean that individuals continuously make sense of the world based on their interactions with it (Saunders et al., 2007). This tradition also holds true for the author interacting with individuals throughout the research leading to the research in itself not being value free. Another interpretivist tradition also followed in this study, is the phenomenological tradition of understanding the way humans make sense of the world (Saunders et al., 2007). This commits the author to consider the context of the respondents as to why they are expressing their meaning in a certain way.

Both of these intellectual traditions perceive the individuals as thinking beings incapable of separating the world they interact with from their own preconceptions of it. As such objective truths are hard to obtain when taking an interpretivist stance. The ontology of this paper is thus subjectivism and focus on understanding the subjective reality that drives users to build certain solutions or govern certain platforms in certain ways (Saunders et al., 2007). Adopting a subjectivist ontology challenges the perceived "truthfulness" of the research as subjective individuals intentionally provide meaning while unintentionally being influenced by bias and interaction with the world. Thus, in convincing readers that the findings in this paper are true coherence theory is adopted, in which it is stated that the conclusions that researchers make throughout a study must be free from contradictions (Saunders et al., 2007).

The interpretivist approach of this study is further supported by Burrell and Morgan's (1979) four paradigms. This model consists of two axes going from objectivism to subjectivism and radical change to regulation (Saunders et al., 2007). The first axis is directly related to the ontology discussion and as established earlier this research follows the principles of subjectivism. The second axis is related to the aim of the research, which can be either radical change or regulation. Radical change aims to criticize and suggest entirely new ways of doing things, whereas regulation aims to describe and suggest new ways of doing things within the existing state of affairs (Saunders et al., 2007). As explained throughout this section the interest of the author is to uncover how the use of low-code development platforms are impacting organizational interests. Throughout this research, knowledge from existing literature is being utilized in identifying sub-optimal ways of working and in suggesting ways of improving these. This points to this research being more towards regulation than radical change. The combination of subjectivism and regulation is placing this paper in the "interpretive paradigm" of Burnell and Morgan (1979)'s model, which is trying to “(…) understand and explain what is going on" (Saunders et al., 2007). This aligns very well with the aim of this paper in understanding how solutions are built by employees and the impact this has on the organization as a whole.

Approach

This research further attempts to adopt an inductive approach in answering the research question as this aligns well with the interpretivist tradition and the interest of the research is to discover the practice rather than testing the theory. The inductive approach as opposed to the deductive approach follows a less structured methodology allowing for alternative explanations than what could have been achieved following a more structured approach. The choice to go with an inductive approach in

this research is further supported by the lack of research on the research topic, which according to Creswell (1994) are the topics best suited for inductive research (Saunders et al., 2007).

As explained previously, interpretivist researchers are also engaging in constant sense-making based on their context making an entirely inductive approach dificult. Knowledge on existing literature introduce a bias to the research, as an initial literature review was conducted prior to designing the research. This inevitably led to a bias resulting in a slightly less inductive approach to the research. The author attempted to account for this bias throughout the research process enforcing the inductive approach on the practices conducted.

During data collection the inductive approach was also predominantly applied. In designing the interview guides prior to the interviews, an effort was made to keep previously mentioned preconceptions from influencing the structure and questions in the Interviewguide. For the use case interviews, the interview guide was slightly biased by the literature as sub-questions based on the literature were included in case the respondent did not elaborate enough (Appendix 1). However, the sub-questions included could arguably be accredited to basic knowledge and logical reasoning rather than theoretical concepts reducing the impact on the inductiveness of the research. During the interviews, however it became dificult for the author to account for this bias which resulted in follow-up questions directly related to the literature (Vicki). For the governance interviews, the literature had no impact on the structure or questions of the interview guide neither on interview. This was primarily due to minimal amount of theoretical concepts on governance had been reviewed at the time of the interviews (Appendix 2).

Data analysis was also predominantly inductive as interviews were coded based on the data rather than a predefined set of concepts (Saunders et al., 2007). Once again the bias impacting the use case interview guides, whether based on concepts in the literature or basic knowledge, was impacting the inductive nature of the data analysis. This resulted in paragraphs from the interview transcripts subconsciously being coded according to this bias. In an attempt to uphold the inductiveness of the research the codes were reviewed, and the paragraphs recoded into codes closer to the data.

In general, this research is attempting to follow an inductive approach with active efforts made to keep the research within this approach. However, the presence of deductive reasonings in the research is undeniable as literature review was conducted prior to data collection and data analysis consequently leading to a deductive bias. As such this research applies a hybrid approach between induction and deduction (Saunders et al., 2007).

Research strategy

A research strategy helps the research fulfill its purpose. The purpose of this research as has been stated many times is to understand how the use of low-code development platforms impact general business interests. As such this research takes an explanatory approach, which concerns the studying of a situation or a problem to explain the relationship between variables (Saunders et al., 2007).

A very common strategy in achieving this is the case study, where emphasis is put on gaining a "(…) rich understanding of the context of the research and the processes being enacted" (Saunders et al., 2007:139). In alignment with the research interest, this strategy is applied in this research, investigating the use of the Microsoft Power Platform by the professional services firm – EY.

According to Yin (2003) case studies can be structured along two discrete dimensions: single case v. multiple cases and holistic case v. embedded cases. The first dimension is concerned with how many cases are being investigated. The second dimension is concerned with whether the research is based on a holistic view of the organization(s) involved or if the subunits of the organization are considered (Saunders et al., 2007). In this paper the use of low-code development platforms are investigated i on one single organization or case and across the entire organization as a whole, pointing to a single and holistic case study. The decision to investigate only a single organization is based on considerations with regards to time. When the decision was made to go with this strategy, the time remaining before the end-date was less than optimal. As such getting access to another case would only increase the risk of not finishing the research on time. The holistic approach was decided as low-code development platforms were not widely adopted within the organization in question and getting access to multiple respondents with experience in working with these platforms and from the same sub-unit was a challenge.

Research choices

It was decided that this research would use a mono-method qualitative approach utilizing in depth interviews as primary method (Saunders et al., 2007). This decision was based on the interpretivist stance taken, where a crucial point is to understand the world of the individuals involved and the combination with quantitative methods, would not add further value to the research. Other qualitative methods could have been included such as document analysis, however the author was legally challenged in obtaining relevant documents. As the author was employed by the case organization, the author was under legal obligation to not share internal documents externally. Other qualitative methods, such as focus group interviews, did not seem applicable to the research interest of this paper.

Time horizon

This research focus on a cross-sectional time horizon partly due to the fixed end-date of the research, challenging the focus on a longitudinal time horizon and partly also due the research interest being of a more static nature. Had the interest of the research been to understand how the use of the platforms and the impact on business interests have developed over time, then the time horizon would have been longitudinal. Furthermore "Many case studies are based on interviews conducted over a short period of time" (Saunders et al., 2007:148), which further supports the focus on a cross-sectional time horizon.

Methods

As has been mentioned previously the primary methods for data collection was Interviews. In selecting respondents for these interviews, it was prioritized to get perspectives from both users of the platforms as well as governing individuals.

A common issue encountered by many researchers is the challenge of gaining access to an appropriate source of the research. This challenge was overcome by utilizing existing contacts, which is a common strategy to overcome the issue and goes very well with the case study approach (Saunders et al., 2007). As the author during this research were employed at the case organization easy access was provided by being on the inside of the organizational boundary.

Data collection use case perspective

In selecting respondents to represent the use cases of the Microsoft Power Platform in EY, self-selection sampling technique was applied in combination with purposive heterogenous sampling (Saunders et al., 2007). The self-selection sampling was done by creating a small survey containing no more than five questions:

1. Have you ever used low-code development tools before?	Choice • Yes • No
1. What tools have you used?	Multiple Choice • Power automate • PowerApps • Blue prism • Ui Path • Other

	○ Free text field
1. How was your experience with software development prior to using these tools?	Choice • I was a developer • I had never been involved in software development before • I had been been involved in software development before • I had managed software projects • Other ○ Free text field
1. Why have you never used such tools before?	Free text field - multiple lines
1. Can I reach out to you for further elaboration?	Choice • Yes • No

The questions were branched meaning that if they answered “no” to question number 1 then they would be directed to question number 4 and 5. In the case they answer “yes” to the first question they would be directed to 2, 3 and then 5. The questions 2 and 3 was added to gain an understanding of the respondents' experience with development and with low-code tools allowing for a purposive sampling later. Question number 4 was added as the author at the time was yet unsure of the direction of the research following an inductive approach. By adding question 4 the author allowed the capturing of possibly relevant cases in terms of users who had not used the low-code development platforms. Question 5 was added for ethical reasons as the author wanted to make it clear that further elaboration was desired and just to make sure that they would agree to that.

The survey was live for a month starting from mid-March to mid-April and was posted twice in organizational communities - once in the beginning and once halfway through. The survey link was posted in three internal communities on “Yammer”, which is a social platform for internal communications provided by Microsoft to organizations. The communities were related to three different topics - Power Automate, PowerApps and RPA. A total of 20 people responded to the survey where one had never used low-code development platforms before. The majority of the respondents had used Microsoft tools before, which might be due to the bias introduced from posting the survey in two Microsoft related communities. A total of 14 use case interviews were conducted of each 30-45 minutes duration. The remaining 6 respondents was purposefully left out of the study. One did not

reply on the authors request to participate in an interview and another had never used the tools before, which was descoped from the purpose of this research after a couple of interviews with respondents who had. The remaining 4 was left out as their email revealed that they were from the same offshore unit in the organization as two of the respondents already interviewed. Furthermore, after having conducted the interview it was decided to limit the scope of the case study to only consider the use of one platform, which resulted in 4 respondent further being descoped. The respondents included in this research representing the use case perspective is presented below:

Sheldon	**Department**: Robotics and Artificial intelligence **Location:** EMEIA - Johannesburg **Working in EY**: 3 years **Role:** Delivering robotics solutions to clients **Background:** Software engineering and business systems and analytics
Aurimas	**Department**: Strategy and Transactions – Transaction dilligence **Location:** EMEIA - Switzerland **Working in EY**: 12 years **Role:** Client serving role focus on data analytics, Internal role coordinating innovation matters in Strategy and Transactions Switzerland and another internal global role in connecting transaction diligence analytics. **Background:** Degree in business and economics, always had interest in code
BMC	**Department**: Core Business Services – Brand, Marketing and Communications **Role:** Building reports and data visualizations **Background:** Bachelors degree in biotechnology and chemistry, later transitioned into Data and analytics
Tim	**Department**: Core Business Services – Nordic Tech Hub **Title**: Associate Solution Architect **Location:** EMEIA - Denmark **Working in EY**: 9 months **Role:** Partly designing solutions partly developing solutions **Background:** Business and IT related
InfoSec (Anonymized)	**Department**: InfoSec – Active Directory Infrastructure **Role:** Managing IAM infrastructure **Background:** Computer engineering and 11 years of IT experience
Vishal	**Department**: TAX – Indirect tax **Title**: Advanced analyst **Role:** Preparing indirect tax returns for clients **Background:** Commerce background, with some experience with SQL and Python
Michael	**Department**: TAX – Tax technology and transformation **Working in EY**: 4 years **Role:** Working with clients to implement the Power Platform and internal process improvements

	Background: Masters in accounting
Parulbahen	**Department**: Core Business Services – Project process implementations **Title**: Data analyst **Role:** Implement old process into new process as a Power Platform developer **Background:** Computer engineering and master in information technology and 12 years as data analyst.
Vicki	**Department**: Asia Pacific Consulting quality team **Location:** Australia **Role:** Ensuring consulting follows all the right procedures and processes **Background:** Degree in international business and experience in project management
Jose	**Department**: Intelligence service **Location:** Peru **Role:** Delivering rapidly development solutions to clients **Background:** Business background

The interviews conducted with the users were semi-structured, meaning that some overall topics and open-ended questions were on the agenda, however the focus was on getting the respondents to talk (Saunders et al., 2007). The decision to conduct semi-structured interviews was partially based on the explanatory nature of the study and partially based on the interpretivist research philosophy. As the semi-structured interview allow for the interviewer to probe respondents to elaborate on their answers, this fits very well with explanatory nature of this study. A semi-structured interview also allows the respondent to bring things to light which they perceive to be important. This is especially important when taking an interpretivist stance as the focus of research within this area is understanding the meanings the respondent apply to the topic they are discussing (Saunders et al., 2007). Prior to the interviews an interview guide was made containing four topics: Introduction, Low-code development, solution governance and final thoughts (Appendix 1). The interviews were conducted using Microsoft Teams, however this did not seem to have an effect as it usually has in the experience of the author. This might be due to the COVID pandemic forcing everyone to work from home making on screen interviews less awkward. The introduction topic was added as it in interpretivist studies is important to understand who the respondent is in order to accurately reflect the meaning they ascribe certain topics (Saunders et al, 2016). The low-code development topic was added as this was the primary nature of the use case interviews and the solution governance topic was added to capture user perspectives on the governance topic. Final thoughts were included to allow the respondent to bring points to the interview not captured by the author throughout the interview. The questions in the interviewguide were biased by the literature review conducted prior to initiating the data collection, which is evident from the sub-questions related to the low-code development topic (Appendix 1). The author attempted to shed this bias by only asking the overall question pertaining to the topic, however in reality the sub-questions had a more significant impact on the course of the interview.

Prior to any interview the respondent would be introduced to the purpose of the research as this would enable them to understand in what direction the interview was intended to be taken. In addition to this introduction, all respondents were asked if the interview could be recorded, if they would like to be anonymous and in case that anonymity was not a concern whether they would have quotes for approval. This was done to follow standards of research ethics entailing that respondents should know what they are agreeing to prior to an interview (Saunders et al., 2007). All interviewees agreed to having the interview recorded, which enabled the author in engaging in more thorough data analysis later. Two respondents opted to be anonymous and five opted for approving the quotes. These answers were noted down in the interview guide for the author to remember to follow up on them later. After this the interview was initiated with the author asking the respondent to introduce themselves. The respondents were not consistent in what information about themselves they introduced so it often happened that the author would ask follow-up questions about their educational background or country of work (BMC; Sheldon).

The author had planned to ask the questions as outlined in the interview guide containing a combination of open-ended questions and probing questions (Saunders et al., 2007). The questions asked in many cases ended up being more probing than open ended such as for instance: "how did you begin using the Microsoft Power Platform so that can be, you know, first power BI I assume you started with that one and then later moved into the other ones correct?" (Aurimas). This introduced a bias as the author put its own assumptions on the respondent leading the respondent to answer in a certain direction. All the respondents were very elaborate in their answers leading the author to sometimes forget the interviewguide and take the interview in other interesting directions. This is a benefit of the semi-structured interview is that it allows for the interviewer to explore topics that emerge throughout the interview (Saunders et al., 2007). This resulted in prolonged interview duration for several respondents as they would have many interesting perspectives on a topic. This was for instance the case with Jose who seemed to have an immense passion for the area of low-code development and started the interview with presenting some slides on the intelligence services team, which he was in the process of building (Jose). In line with the interpretivist research philosophy the author allowed for such a deviation from the Interviewguide (Saunders et al., 2007). In the end the author had to interrupt and ask some of the questions from the interviewguide to make his case comparable to the other respondents.

Data collection governance perspective

In selecting respondents to represent the governance perspective of the case a combination of purposive and snowball sampling within the overall non-probability sampling was applied (Saunders

et al., 2007). The purposive nature of the applied sampling technique comes from the identification of a product manager in charge of the Microsoft Power Platform within EY. Based on the first interview with this respondent, the snowball sampling technique was utilized in identifying other influencing parties on the governance of the platform. This revealed two other stakeholders - one with similar interests as the original respondent and one with different interests. An overview of the respondents is provided below:

Eoin	**Department**: EY technology – End User technology Experience **Title**: Product manager **Responsibilities**: Looking after the Power Platform **Working in EY**: 2-3 years **IT experience**: 15-16 years in general IT
Anbu	**Department**: Client technology – low-code services **Title**: Leader (Manager) **Responsibilities**: Partnering with service lines in building solutions on top of Office 365 dynamics or Power Platform **Working in EY**: 2 years **IT experience**: 20 years with Microsoft technologies
Security consultant (Anonymized)	**Department**: Security consulting - end user computing **Title**: Platform consultant **Responsibilities**: Review and assess risk on platforms that are specific to internal end users. **Working in EY**: **IT experience**: Some software engineering experience focusing on back-end infrastructure

The product manager was interviewed twice for 30 minutes each time and the stakeholders were each interviewed once also for 30 minutes. The interviewguide used for these interviews was originally designed for the product manager containing five topics: Introduction, Low-code development platforms, Governance of low-code development platforms, Processes pertained to low-code development platforms and Final thoughts (Appendix 2). The introduction and final thoughts were included for the same reasons as with the use cases. The low-code development platforms topic was aimed at capturing governance perspectives on the platforms in general, allowing the respondent to bring their individual opinion on them to the table. The governance of the low-code development platforms topic was aimed at capturing actual governance mechanisms surrounding the platform. Finally, the processes pertained to low-code development platforms were aimed at capturing the more detailed processes related to the platforms. This interviewguide was reused for the stakeholders, as they were involved in a lot of the same processes as the product manager. One exception was that the

topic related to specific processes was taken out, as it was expected that these respondents would not have any input on this. This interviewguide was not biased by the literature review as this area had not yet been reviewed by the author.

As with the use case interviews, Microsoft Teams were used in conducting the interviews. The respondents were also here introduced to the topic prior to interview start along with the formality questions concerning recording, anonymity, and quote approval. One respondent opted to be anonymous here and the others opted for quote approval. The approach in the interviews followed the same structure and format as the use case interviews.

Data analysis

In analyzing the data, the thematic analysis framework by Braun & Clarke (2006) was followed. This meant addressing the analysis through the six-step guide provided in their article.

The first step of this framework is "Familiarizing yourself with your data" (Braun & Clarke, 2006:87). This was done as suggested by them through transcription of the verbal data obtained through the interview recordings. In transcribing the interviews, it was decided to transcribe exactly what was said rather than transcribing for meaning. This was done as the inductive nature of this study meant that the author did not know what would be relevant at the time and the risk of losing important nuances in translation were considered significant. A common challenge experienced in this process was the dificulty of understanding what is being said due to poor internet connection or differences in accents in the English language.

Next step entails "Generating initial codes" (Braun & Clarke, 2006:88). Coding was done in Nvivo allowing for nice organization of multiple files and codes. As described previously an inductive approach was adopted in the coding, which led to 200+ codes. This was partially due to the author's lacking experience in conducting inductive coding and thus assessing the right level of detail was difficult. The author was biased by initial literature review when creating the initial codes, which resulted in codes such as "requirements gathering" and "testing solution". These biased codes were to stay true to the inductive nature of the study recoded more closer to the data. Furthermore, the lacking experience of the author in this type of coding resulted in codes with +60 references such as "Governance", which were also recoded. After having recoded a these the total codes resulted in +400 codes.

Based on these codes the next step "Searching for themes" was initiated (Braun & Clarke, 2006:89). This was done by organizing the codes in new higher-level codes based on themes discovered in the

data. These themes include "Internal formal use", "Internal informal use" and "External formal use". The codes that did not seem to fit into a theme was collected under a theme called "Miscellaneous", which according to Braun & Clarke (2006) is common following an inductive coding approach.

Having identified key themes in the data the next step was initiated: "Reviewing themes" (Braun & Clarke, 2006:89). This step entailed reviewing the themes on two levels, first within the theme itself and then across the entire dataset. Within the theme itself inconsistent paragraphs were recoded to another theme, ensuring that all codes under that theme actually were related to that theme. In the second phase the themes identified were considered as to whether they were accurately representing the dataset as a whole. In this case a theme related to "Upskilling" were moved to miscellaneous as this theme did not accurately reflect the dataset.

After step four the framework was not followed any longer as the rest were regarding the naming of the themes and writing of the report, which was a natural next step of the research. The themes were however reviewed and amended throughout the process of writing the report as the author discovered new topics within the themes. This resulted in a break down of the themes in smaller sub-themes.

Case description

EY formerly known as Ernst & Young is a professional services corporation, who is globally present with it's 300,000 employees worldwide serving 200,000 clients in 150 countries (EY, n.d.-b; EY, n.d.-a). Their purpose is defined as:

"At EY, our purpose is *Building a better working world*. The insights and quality services we provide help build trust and confidence in the capital markets and in economies the world over. We develop outstanding leaders who team to deliver on our promises to all our stakeholders. In so doing, we play a critical role in building a better working world for our people, for our clients and for our communities" (EY, n.d.-c)

In fulfilling that purpose, they value: "People who demonstrate Integrity, respect teaming and inclusiveness, People with energy enthusiasm and the courage to lead and People who build relationships based on doing things right" (EY, n.d.-c).

And their vision is: "We believe a better working world is one where everyone can contribute to and share in the benefits of sustainable economic growth" (EY, n.d.-c)

The organization is structured in the executive and regions. The executive entails the global leadership, governance bodies and three geographic Areas - Americas; Europe, Middle East, India and Africa (EMEIA); and Asia-Pacific. The executive is working together to "(…) oversee our global strategy, brand, business planning, investments and priorities (…)" (EY, n.d.-a). The regions consist

of 28 separate regions grouped under the overall areas (EY, n.d.-a). Their services are divided in four client serving service lines – Assurance, Tax, Consulting and Strategy and Transactions – and the Core Business Services as their internal service line (EY, n.d.-b; Parulbahen).

In 2018 EY globally decided to upgrade their Microsoft office 365 suite from the basic package containing standard applications such as SharePoint, Outlook, Skype and OneDrive for business to the extended package including the Power Platform (Eoin 1). The Power Platform entails Power BI for data visualization, Power Apps for application building, Power Automate for workflow automation and Power Virtual Agents for chatbot configuration (Microsoft, n.d.). This upgrade is available to every employee in EY and costs hundreds of millions of dollars every year in licenses to Microsoft (Eoin 1; Anbu).

Analysis

EY's use of the Power Platform

This section will analyze how EY is using the Power Platform. Three use case scenarios were identified through the data analysis moving across two dimensions – Organizational scope (External/Internal) and formality of agreement (Formal/Informal). This section will describe and how these three scenarios are working with the Power Platform followed by an analysis of the impacts these approaches have based on software engineering best practices as outlined by Sommerville (2015).

External-Formal scenario

In this scenario the Power Platform is utilized as an asset in delivering successful projects to clients: "We're trying to make this new service focusing intelligence services. We're not trying to sell low-code because low-code is a concept" (Jose) and "I've kind of been working on the, you know directly with clients on how to implement the Power Platform for different processes that they have" (Michael). The overarching goal for these teams, being client serving teams, is thus to increase sales of that specific team by using this new type of technology. The projects can vary in nature from implementing a COVID care application for a farming company in Peru (Jose), to building a Power Platform Center of Excellence (CoE) with the African clients (Sheldon). Sheldon keeps emphasizing the interest to not embed EY with the client, but instead focus on helping them in building the capability themselves. The Jose do not seem to share this approach, but rather they deliver solutions with the platform to clients and implement them on their environment (Jose). Besides these actual use cases the interview with the manager of low-code services within Client Technology revealed another type of project: "we will be starting of a product development with one of our service lines (…) and build a product right on top of Power Platform and they sell the product to the clients" (Anbu). This type of project still has the aim of increasing sales with clients, but the approach is different as the

product is developed in house and then later sold to the clients. The way these solutions are delivered vary between client implementations and managed service: "we're just onboarding client users onto our environment so that is like 80% of the use case. 20%, we build a product. And we deploy to the client's environment" (Anbu). Client implementations resemble what is done in the other client serving teams, but the managed services approach is quite different. All teams do however seem to agree that the Power Platform can serve as valuable asset in increasing income from clients.

When working on client projects gathering requirements up front is hard to avoid as they " (…) can be a bit fixed purely because we require that client sign off because we need to manage scope that way" (Sheldon). Consulting Peru follow a similar process: "(…) first of all we define methodology (…) we identify the new requirements in meetings with the final user that we call the SMEs the subject matter expert (…)" (Jose). They define what methodology to follow first, and independently of the choice they identify the requirements up front. Sheldon further expresses that getting the client's sign-off is a part of EY's internal processes to start an engagement with a client. Sheldon explains how they cope with this: "*what will normally do is we'll have an overarching waterfall approach, but every single requirement that we develop until later on in the build phase will be done very iteratively*" (Sheldon). The nature of these requirements is unclear, but the combination of waterfall and agile delivery methods implies quite high-level requirements. To iterate on any one requirement they need to be defined on a relatively high-level. Jose on the other hand exemplifies the requirements stating that they for example would "(…) get the details about the functionalities, about the rules and about the volume of the transaction (…)" (Jose). Getting details contradicts the argument of high-level requirements, however details could refer to the initial documents needed prior to initiate building such as "(…) process flow, (…) mockups about the front end (…) documentation about the rules and some technical information" (Jose). In this case a similar process is followed in Sheldon's team: "We'll do an analysis phase, will see what the requirements are, gather requirements, the sign off on the requirements" (Sheldon). Both teams seem to focus on gathering proper requirements prior to design of the solution.

The design of the solution in both teams also follow a similar process. In Sheldon's team they will do a design sprint focused on understanding the requirements more in detail along with the required infrastructure and resources (Sheldon). In Jose's team they spend the first sprint identifying "(…) the elements of the new functionality and the development userstories" (Jose). They both focus on further specifying the requirements during the first sprint, however Jose mentions the development of user stories, whereas Sheldon only mentions understanding the requirements, infrastructure and resources. Furthermore, they in Jose's team work with a technical team to identify and make the design of the solution, however whether this is before or during the design sprint is unclear (Jose). The involvement

of a technical team is a bit different than what Sheldon's team is doing, which could be accredited the fact that Sheldon's team primarily consist of software engineers (Sheldon).

Both teams have a clear understanding of what needs to be developed after the first sprint. Based on those requirements the teams will start development: “(…) we have the other sprints, development [and] quality of the release one (…) development and quality of release two” (Jose) and “(…) we'll go into build and pretty much every two weeks we'll try and release (…) and we'll have generally a feedback workshop as well with our stakeholders (…)” (Sheldon). Both teams utilize iterations or sprints to deliver parts of the solution and have some sort of quality assurance prior to moving into the next sprint. In approaching the development both teams also seem to adopt the concept of an MVP: "We have a (…) specialist in the business, in order to define the step-by-step process to identify what the main functionality is for this MVP” (Jose). An MVP is a "Minimum Viable Product”, and it is as Jose states a product containing the main functionality. The concept of MVP's was coined by Eric Ries in his book "The lean startup", where he tells about his experiences and thoughts on how to best build successful businesses or products (Ries, 2011). Sheldon capture the main ideas quite well: "We very much adopt let's build MVP's, let's fail fast, let's iterate on the product as many times as we need to get it right (…)” (Sheldon). Both mention delivering projects faster than they usually were able to as the low-code platform require less manual work (Sheldon; Jose).

As described previously each sprint will in both teams contain an element of quality assurance or testing of the system. The approaches of the two teams are very similar and contain various types of testing. One approach they both utilize is testing with the individual developer: “(…) the developer will do their unit testing in the development environment (…)” (Sheldon) and “(…) the developer is responsible to code and test (…) the component” (Jose). This might not come as a surprise to the average software engineer as it is very logical to test out the code during development. To ensure this testing does not impact other applications or production data, Sheldon and his team " (…) will mimic what the production environment would look like with test accounts, test data, test users and [in] that way we can be in complete control of all the moving parts" (Sheldon). This approach prevents the spamming of people as Sheldon explains (Sheldon). This is not something that Jose mentions, however since they are launching solutions to the client’s environment it is fair to assume that client data is separated from the development environment.

Besides the unit testing Sheldon's team make use of another type of testing: “(…) we'll test it out, put it into what we call a soft go-live state where we get some beta testers in” (Sheldon). As described in the literature review, beta testing entail allowing a small user group early access to the solution. This approach is slightly different in Jose's team: “(…) and after that [developer testing] we have integration integral testing with the tester, it's on is another role” (Jose). Jose's team also do testing

after having developed and tested individual pieces of code, however they utilize an actual tester rather than the users to test the code before deployment. Both teams add a third type of testing to their process: "After that we have the UAT, the User Acceptance Test" (Jose) and "(…) UAT, and end to end test cycle once we've kind of completed all of our build sprints (…)" (Sheldon). In relation to testing in the Power Platform Sheldon explains that it is largely manual testing, but they do sometimes "(…) train up Ui path or blue prism robots to imitate users and so that gives us high frequency and high volumes (…)" (Sheldon). This is something only Sheldon mentions, which could be due to the robotics team being used to working in those tools enabling them to spot such an opportunity. However, this is only done for high-volume testing. Sheldon explains that at the UAT marks the end of the project where the client signs-off on the solution and they start thinking about how to get the solution into production (Sheldon).

When putting the solution into production both teams offer to maintain and support on a client solution for a limited timeperiod, however this requires a formal agreement (Jose; Sheldon). Sheldon states directly that they do not want to keep maintaining the solutions as they prefer being a build capacity and not embed themselves at the client (Sheldon). This also explains why the projects Sheldon's team does has the long-term focus on building a CoE with the client. Furthermore, both teams are open to work on the solution after delivery (Sheldon; Jose). Sheldon explains that this would require the client to provide new requirements: "If there's any specific requirements that you do need in the future, cool come back to us as an advisor and will work in that capacity" (Sheldon). None of them explicitly mention going through the process again, however new functionality and new requirements are expected to be scoped as with any new software delivery. Jose further explains the process they have of implementing changes: "(…) based on the requirement we review it with the technical team, (…) [if] there is another better way to do that or there is more easy way to develop it" (Jose). This type of review is not mentioned by Sheldon, which could be due to the technical capabilities of his team and thus they just review requirements by themselves.

The use of the Power Platform to deliver client value is partly done through the development of a COE with the client and partly though the actual delivery of solutions. Both teams have a formalized and structured approach in taking client requirements, translating them to a solution and ensuring that the solutions is of the right quality. The Power Platform allows them to deliver working solutions much faster than previously possible. In the end maintenance is preferred to be with the client as they are mor interested in serving as an advisor.

Internal-Formal scenario

In this scenario the Power Platform is used to optimize and automate internal processes in the broader organization: "So this team I'm working in (…) has the purpose of automating digitizing EY from the inside (…)" (Tim) and "Project process implementations (…) we implement old process into the new process and for the Internal stakeholder" (Parulbahen). The overarching goals of these teams are cost reduction and digitalization, as they target existing manual processes and use the Power Platform to optimize, digitize and automate these. The projects worked on in these teams can vary but usually they revolve around an internal pain point such as "every year there is a whole process in place which was done manually (…)" (Tim). Both teams developed a solution involving Power Apps as a user frontend, Power Automate as backend and Power BI for reporting (Tim; Parulbahen).

The Nordic Tech Hub seem to have been following a structured approach to requirements gathering: "(…) the business stakeholders and (…) the project manager were already in discussions before (…) But once I joined, we tried to take it from the beginning again and go through the process" (Tim). Through interviews with the stakeholders the process is walked through in a story like fashion. Similarly, the project process implementation team also had the project manager gather requirements: "(…) project manager gather all the requirements from the team and then he discussed with me everything" (Parulbahen). The two approaches differ in that Tim seems to be included in meetings with the internal client, whereas Parulbahen is only engaging with the project manager. The requirements initially gathered by Tim's team seem to be quite high-level: "(…) in the beginning we didn't know what technology we were going to use (…) the actual system requirements would first came later (…)" (Tim). This suggest that details or the technologies of the solution had not been discussed, which suggest rather high-level requirements. The nature of the requirements in Parulbahen's team is less clear, however she explains that "The project manager handle everything and we discuss if a feature is available (...)" (Parulbahen). The use of the word feature to describe the requirements suggest a more thoroughly defined requirement, which might or might not be possible in the solution.
Tim mentions producing documentation in the early stages of the project: "(…) process diagram (…) solution architecture (…) user stories" (Tim). These documents help Tim's team gain a deeper understanding of what needs to be developed. For the project process implementation team documentation is also created although not mentioned directly: "(…) he sent me the draft of how it will look like and then some of the (…) work we need to be automated" (Parulbahen). This draft seems contain both design mock-ups as well as some logic behind the application helping the developer understand exactly what needs to be developed. In both cases these documents seem to have been created prior to initiation of development.
Having developed documentation on the solution, Tim's team would show it to the client: "Then we built the solution architecture and talked a lot with them about that" (Tim). By showing the solution

architecture to the client, they are invited to provide feedback, prior to the initiation of development. In the project process implementation team "(…) the requirement comes through and then I [Parulbahen] need to research about it. (…)" (Parulbahen). As described previously these requirements also include design of the solution, which indicate that the project manager engage in discussions with the client without Parulbahen. When she receives these requirements research is done on what is possible. Research on what is possible was also done in Tim's team:

> "(…) we built a simplified database in order to test that POC (…) in PowerApps, using actual data from the stakeholders. And it didn't look pretty it it was purely to see. OK, can we actually do what we want to do in this scenario" (Tim).

Rather than researching online documentation on what can be done, the Nordic Tech Hub built a proof of concept (POC) using the actual data to test their assumptions out. Parulbahen would based on her research provide feedback to the project manager: "(…) I mostly send him [the project manager] on the my feedbacks, opinions, what it will look, (…) how it's going to work out by the end users" (Parulbahen). It is fair to assume that the project manager based on that feedback will go back to the client and present the details on which the client can comment. In both cases a sanity check of the solution is done prior to development. However, in Tim's team they will define the solution and present it to the client, whereas in Parulbahens team they would check if the clients' requirements are possible and provide feedback if they are not (Tim; Parulbahen).

In Tim's team the client would accept on the POC, acting as a sort of informal sign off from the client (Tim). In Parulbahens team nothing is mentioned about the client having to accept a certain solution, however since the requirements come very detailed from the client it could be interpreted as the accept from the client. In Tim's team they would continue with some more detailed solution design: "(…) We spend more time to revisit the database and. See again OK, what is it actually? We need to solve the requirements and at the same time we had a UX designer come in to see how should it look like" (Tim). This detailed design includes looking at the database as well as defining how the user experience will look like. This design was in Parulbahens team already defined by the project manager: "(…) he will design everything and then my part will be the development of that design (…)" (Parulbahen). As the requirements are very detailed up front by the project manager, Parulbahen only needs to concern herself with the actual development of the solution.

During the actual development, the Nordic Tech Hub "(…) spent very little time on documenting once we started building(…)" (Tim). Tim explains this with a lot of changes coming in during development and they tried to "(…) build it and move on to the next thing as fast as possible (…)" (Tim). This worked a little bit different for the Parulbahens team: "(…) while I was working on it, I need to test everything and then new requirement was like little bit little bit was coming (…)" (Parulbahen). In their case the requirements from the client would come incrementally throughout the

project rather than actual changes. Requirements are thoroughly defined by the project manager as explained earlier, leading to a little more documentation in this team.
The Tim's team would work in sprints, having daily stand-up meetings on progress and utilize userstories in delivering requirements (Tim). Parulbahens team seem to just have one developer working on the requirements from the project manager, with no formal structure defined for the actual development (Parulbahen). Tim explains how the constant changes affected their ways of working: "(..) we kind of lost track of some of the items and (…) didn't really plan and move in sprints anymore (…) We constantly were showing it to people and (…) we still had weekly calls with the business stakeholders in order to get their input (…)" (Tim)
This suggest that Tim's team had weekly calls with the client to get input and feedback on the solution throughout development, however due to the constant change and lacking documentation they lost track of the items and loosened up their structure. At one point they even had a workshop with the partners using the app: "(…) they had a lot of different ideas of what they would like to see and how it should be presented to them and what they have to change (…)" (Tim). Tim explains that their initial assumptions were far from what the partners wanted, and a lot of rework had to be done based on this. Parulbahens team also experienced changes throughout the development besides the new requirements: " (…) I need to change in this one the layout and everything I need to fix it properly that everything comes across in one form" (Parulbahen). These changes seem to be a result of the new requirements received throughout the sprint and thus the change is conducted concurrently with implementation of the new requirements.

Tim's team tested the software during development in a couple of ways. First of all, the developer would conduct small tests of their code: "(…) when I was developing something I was of course also doing small unit tests in order to see does this work the way I think it should work (…)" (Tim). This was also done in the project process implementations team: "(…) while I was working on it, I need to test everything (…)" (Parulbahen). In both cases the individual developer would test to ensure that their developed piece of software was working as expected.
When the developer in the Nordic Tech Hub was certain that the piece of code was working, a testing resource would take over: "(…) once it works, I was able to hand it over for more elaborate testing (…)" (Tim). Prior to the development, Tim and the tester would sit down and define all the test cases: "(…) when we had developed all the user stories (…) testing resource and I actually spent time together in order to build test cases from the user stories" (Tim). These test cases would make up the foundation of the testers work to do elaborate testing throughout development. As explained earlier changes throughout the project were not properly documented, which challenged the handover to the tester: "(…) we had to talk a lot back and forth because some information wasn't properly documented, so we had to take more meetings compared to how it could have been" (Tim). This lacking documentation further challenged the team in their overview of what had and what had not

been tested through the development: "(…) it was difficult after while in order to understand which ones which were revisited after they had been closed or actually tested already (…)" (Tim). As the test cases was not reset after conducting a change on a piece of software, the team lost track of the tests conducted.

Parulbahens team was less elaborate in their testing as it primarily was the developer testing throughout the entire project. However, Parulbahen were being a bit more elaborate in her developer testing: "I have a couple of few not test environment but test account so I used them as a testing scenario so I logged in as them on my computer (…) [to see] how the end user will see (…)" (Parulbahen). Having some test accounts available allowed her to not only check that the piece of code was working, but also how it was experienced from an end-user perspective. Parulbahen further mentions having a test site with dummy data and that the test accounts have all the Office 365 licenses allowing her to test the user experience end to end throughout development (Parulbahen). After having tested everything on her end she would: "(…) create same type of flow or maybe the copy and import that flow in a different account" (Parulbahen). The launch of the solution in a different account allows her to separate the development and testing from the production data. This separation is something that is also done in Tim's team: "(…)we will then keep deploying the app into production (…)" (Tim). This is meant to prevent partially working software from impacting the production data.

Once all the pieces of code have been tested and the solution was ready to go live the users themselves were asked to test: "(…) we had a small group And that was six individuals and what we tried to get some more feedback from them (…) like when the big crowd comes that everything is already fully in place" (Tim). A small group of six people was allowed early access to test the solution and provide initial feedback. Doing a final user testing was also done in Parulbahens team: "Once everything was OK, all the implemented. Then we wanted to test by the proper user (…)" (Parulbahen). By proper user she refers to someone from payroll, which her project manager pulled for testing purposes. In both cases this test resulted in some minor changes to the design or functionality of the solution (Tim; Parulbahen).

After go-live the Tim's team continue working on the solution: "So while we were live I was still developing further stuff further. Like that, suddenly the and another type of user had to be able to see the same information, so I had to build a new screen for them (…)" (Tim). Tim does not dive into how this was decided, however by using the word suddenly he insinuates that this was an unexpected extra functionality. Parulbahens team also implemented some minor changes after go-live: "(…) after testing and after getting live also I did little bit change (…) of the wording or some extra features need to add so it's just maybe one or two new things needed to add but that that's it" (Parulbahen). It is not clear who proposed the changes, however changing the wording would probably be something the

client would ask for. In both cases the changes seem to have been done without question or analysis in the early stages of the solution launch.

Another issue that occurred after go-live in Tim's team, was the need to change data in the database, which originally was not a feature of the solution. In the beginning these changes were handled by the developer manually submitting the changes directly in the production database, however "(…) after the third time of someone having the exact same very valid request (…) I decided to rather spend some more time on (…) give them the possibility to look at all those things themselves" (Tim). Tim decided to improve the solution even further to reduce maintenance and improve solution performance. In a similar way Parulbahens team improved their solution after launch: "(…) after few months I learned that I can create the flow using office 365 connector in a different way (…) so I created flow again using all the extension attributes" (Parulbahen). This statement insinuates that even though she had created the flow and it worked fine, she went back and optimized it once she learned of a smarter way to do it.

The decision to add new functionality was done without question in the initial time after launch or "hypercare" as Tim calls it (Tim). However, during this period more elaborate feedback from the users as well as the developer's own feedback was noted down for later development. Tim explains that they will have "another session where I will focus on revisiting some of the different topics and building on it again during the summer" (Tim). In the Project Process Implementations team there is no plan for further development. Instead, they have a process for future requirements: "(…) if any new (…) change requests come along (…) it's depends who is the project manager who is handling on what" (Parulbahen). Parulbahen indicate that there is a change request process, where the changes are scoped and assigned to a resource. However it is not necessarily the same resource as who developed the solution that will do further development on the solution.
As both teams develop solutions for internal clients and there is no hand-over of maintenance, the teams are challenged by on the one hand doing maintenance on existing solutions and on the other building new solutions. This is a challenge Tim also describe: "(…) there's always this balance that we as a team have as well that one hand we go build the next big thing, but at the other hand we also have to maintain what we already built and make sure that those things work the way they should" (Tim).

The use of the Power Platform to optimize and digitize EY follows a seemingly structured approach. Requirements of the clients are formally taken, however as the solutions are developed these requirements seem to be less representative of the solution. Some quality assurance is made on the solution, however constant changes from the internal clients challenges the tracking of the testing

efforts. In the end the developers of the solutions are also the ones maintaining them as no formal handover is occurring.

Internal-Informal scenario

The internal-Informal scenario usually utilize the Power Platform to optimize own work or the work of the team they are part of. These use cases exist all over the company both in client serving service lines: "I've actually started using it just to automate my own work a lot of the time (…)" (Sheldon) as well as the internal service lines: "(…) I started to use power automate to create different types of flow for my team (…)" (Parulbahen). The overall purpose of these use cases are focused at relieving inconveniences of themselves or the team they are part of, resulting from lacking overview or manual work. Some use cases revolve around "(…) power BI reports that report on the status of your engagement" (Michael), while others relate to "(…) a form submission and then using power automate to email the end results (…)" (BMC). Most of the use cases in this study are informal of nature, focusing on solving own or team problems.

Quite diseparate from the other scenarios requirements gathering is rather informal: "(…) I knew exactly what I wanted to achieve (…)" (Aurimas) and "I just kind of, you know, knew (…)what the process was, and what I needed to put in place" (Vicki). Quite on par with the informality of the use cases, no formal requirements were gathered as the developers already knew themselves what the process was. The decision to improve a certain process are often self-initiated: " (…) I want to automate that particular process, so (…) I went to the SharePoint team for a solution and they said that is not possible with them(…)" (InfoSec 1) and "I actually sort of did it a bit on my own initiative (…)" (Aurimas). As these solutions are initiated on own initiative, the requirements are already known to the developer and thus requirements gathering is completely absent from the process.

Sometimes the developer is developing something for their team and in those cases some requirements are taken: "(…) a lot of it was evolved from actually liaising with the actual end users and making sure that their you know their feedback was incorporated" (BMC) and "my team if I say they are the actual users of this solution, so I discussed with them and I collected their inputs (…)" (InfoSec 1). However, requirements seem to be taken concurrently throughout development of the solution. One person also collected requirements for a solution that would impact the work of others outside his team:

> "(…) If you're the approver and stuff like that and so in those cases it's I mean you could say yes there was an element of collecting requirements, but it's sort of informal conversations with, let's say a service line leader" (Aurimas).

What they all have in common is the involvement of end-user opinions in the development of the solution, however, this seems to be done in a continuous feedback loop rather than actual requirement specification. These more team focused projects are sometimes initiated by a team member: "I actually went to her and kind of pitched this idea (…)" (Michael). These are very similar to the entirely self-initiated with the exception that the more team focused projects were discussed with a manager. Other times these projects are initiated by a broader team: "(…) the team (…) came up with these kind of ideas and you know how we can simplify things" (Vicki). This example indicate a need or decision from the team to engage with utilizing the Power Platform. In all cases the requirements guiding the developing efforts could be either a certain process, whether specified "(…) me taking a sheet of paper and planning it out for myself(…)" (Aurimas) or not: "(…)I was one of the users right, so I know how these things are working(…)" (InfoSec 1). In some cases an old process artifact such an existing excel spreadsheet could also serve as requirements: "(…) basically just following up a an old data entry template and then just replicating that within power apps" (BMC). Either way the whole requirements process is rather informal.

The development process in the scenario seem to continue the informality pattern from the requirements gathering and not worry too much about designing the solution prior to development. This also accounts for the actual development efforts: " I wouldn't say there was any. I guess formal structure that we followed like any sort of, you know, like you know best practices and stuff like that (…)" (BMC) and "(…) just started playing around with it, I guess" (Vicki). As no design documents is produced or planning is made prior to the development this lead to a form of "trial and error" approach where the design is developed as the solution is built. Factors impacting the composition of the solution vary between the respondents.

A few of the respondents mention available tools in EY as an influence when developing the solution: "(…) I've never had proper infrastructure in the sense of like SQL servers or anything like that always was a bit restricted with what's available for free in EY (…)" (Aurimas) and "(…) thankfully you know EY have I guess the SharePoint's as a way of storing the data (…)" (BMC). As Tim expressed during his interview getting a SQL server can be a composome process, whereas SharePoint is available to everyone and therefore in some cases both formal and informal a popular data storage solution.

In some cases the solution design is inspired by existing templates available on the platform: "(…) I took a look at a few of them [The templates], got some ideas, and then I actually sort of started a flow from scratch (…)" (Vicki) and "(…) I started looking for quality templates available on the automate and then I using one of the template (…)" (Vishal). This approach focus on reusing designs from existing solutions, which relates pretty well to the fact that no design was made prior to development.

User experience was also mentioned as a big factor impacting the composition and design of the solution: "I kind of always, you know, kept the user experience in mind (…) [where] the Power Platform sort of comes together is that you can integrate everything with everything (…) " (Aurimas) and "(…) we cannot use some user good user experience using the power automate (…) I'm trying to implement using their power automate. Sorry power apps" (InfoSec 1). The user experience would in both cases impact how the solution was composed with various other applications and the Power Platform allows for easy composition of Microsoft tools. From a user interface perspective, both of the respondents mention the power apps tool as a good solution to achieve that.: "(…) and you know there [in PowerApps] you had a tablet or a mobile version. It's really just working with the UI there (…)" (Aurimas) and "I want to make sure that this should be accessible and in like multiple devices, whether it is in tablet or phone" (InfoSec 1).

One respondent utilizes variables to abstract some of the inputs that are reused across an entire flow: "(…) all I had to do is select the drop down to change one variable, that would then kind of run through my flow as opposed to changing each source connector within each step that used it" (Michael). This technique was introduced to his flow based on the inconvenience of changing data sources when moving from test to production.

Another respondent puts in extra data capture points in his flow: "(…) having multiple just capture points of that data to prevent you know any any loss (…)" (BMC). As he states this was done to prevent any data loss, but the original goal was really to send the responses in an email.

Simplicity of the solutions also seem to be a contributing factor: "we're just trying to keep it nice just as simple as possible." (Vicki), "(…) making sure that they are always working and maybe even simple enough (…)" (Michael). All of these different factors influence how the solutions are designed.

For the actual implementation these use cases follow an informal iterative process: "(…) so we had sometimes iterations where, let's say the account role app. We first include the certain roles and then it turned out that teams maintain more roles themselves in Excel (…)" (Aurimas). The use of iterations in that context suggest that the product is being developed incrementally, however no formal process is followed. Other respondents mention liaising with users or talking to their team to get feedback (BMC; Vicki; Vishal; InfoSec 1), which suggest a similar iterative approach. The size of the automation also influences these projects because as "(…) it would be like a day or even less of work to try to set up the initial thing, test it out, maybe come across something (…) [and] spend a bit more time improving that" (Aurimas).

Testing in this scenario is largely unstructured "(…) the only important thing was to make sure that it works for everybody who for whom it needs to work and there were a couple of times I had to ask other people to try it for me" (Aurimas). This indicates that no structured plan was followed in the testing and for end user testing he would just reach out and ask other people to try it out. Most of the

developers would also do some testing by themselves: "Well, I create a test item I put myself as the reviewer. I start trying to different, you know, status updates and changes (…)" (Michael) and "the initial testing was just sort of, I mean myself triggering the flow and testing to see how it works and whether it works (…)" (Vicki). This type of testing resembles that of the other two scenarios and is a commonly used approach. In addition to this, the developers would sometimes pretend they were the users: " (…) the user testing part of it was. I guess you know predominantly conducted by myself and pretending that I was the end user (…)" (BMC) and "(…) you put together the app you either yourself, make sure to go through all the paths (…) and see if they find anything that doesn't work the way it should" (Aurimas). This type of testing allows the developer to focus on the user experience as was mentioned by most of the respondents.

In all cases testing involving the actual users is done, however the approaches to this activity vary between the cases. For the most part user testing is done by onboarding a small user group prematurely: "(…) I given the access to the few of my colleagues to the test environment and we tested" (InfoSec 1) and "(…) for testing purpose I shared with you I gave access to only a few of my teammates and I asked them to just to enter the details and to check (…)" (Vishal). This type of testing is also used by other respondents and it allows the developer to catch some details of the application prior to full scale release (Michael; Vicki;). Both Michael and Vicki refer to their testing as gradual deployment, where the user group grows as the solution is proven in practice. This could partially be due to the end-user group often being rather small and within their own team, leading to a premature launch having minimum impact. In those cases feedback is incorporated through the further maintenance of the solution: "(…) the campaign teams themselves (…) they were quite happy to, you know, provide their feedback and do a lot of that testing themselves" (BMC). This is more an example of an actual release and continuous improvement over time. The less structured approach to testing could be due to the size of the solutions: "There were some apps which were a bit more complicated. (…) these requires more time from the perspective of making sure it works properly and then there were some others where the testing was probably 15 minutes of work" (Aurimas). Other respondent also insinuate the simplicity of their testing activities when concerning a simple automated workflow (BMC; Michael; Vicki).

Most of the respondents in the internal informal use case does all the testing in the live environment but keep control by only onboarding a couple at the time. Sometimes the data underlying the solution is changed for testing purposes: "(…) it's linked to a SharePoint list which I've set up with the appropriate approvers so (…) during the testing rounds, I've been setting that up appropriately, for whoever's testing" (Vicki). This is done in the same environment as the solution is supposed to run in. One of the respondents attempted to separate development from production: "I actually set up a mimic the environment of the SharePoint site and the library you know just had the suffix of test at the end of it. Yeah, and I was able to literally just have a development versus a production environment for

my testing (...)" (Michael) This is an approach that is recognized by the formal teams in a lack of better options.

Once the solutions have been developed and deployed the maintenance is kept with the developer or the team: "(…) it's kind of on me to. Just make sure everything is working (…)" (Michael) and "They are maintained by me or I have now some people in our offshore centers people who are sort of partially familiar with the setup" (Aurimas). The time it takes to maintain these automated solutions would thus be going from other business tasks of the developers. However, maintenance of the solution seem to be of little concern to the developers: "(…) just didn't seem to break at all" (BMC), "(…) I never notice any fails so far. It is running more than a year now" (InfoSec 1) and " (…) there's some legacy flows that I still have running that had just worked for two years now (…)" (Michael). It seems that the Power Platform built solutions are relatively stable in their performance and thus maintenance is not a big concern for the most part. However it does sometime happen that a solution fails or breaks down: "So there sometimes we get into errors, so we try to then try to do brainstorming what would happen a few of my team members (…)" (Vishal). This approach is involving the other members of a team in a brainstorming session to try and figure out how to fix it. Another scenario is that a flow "(…) suddenly it fails out your flow and then you have to go troubleshoot, figure out where it failed in the flow and then either correct from the source or correct within your flow to make sure it's more dynamic" (Michael). By improving the flow, future errors of that kind is prevented, essentially increasing the stability of the solution, whereas correcting in the data source would only fix that specific error. A common issue that a couple of respondents point to is the difficulty of debugging flows in power automate: "There are some hidden properties in certain of the whatever level tools this call it right that you need to figure out" (Aurimas). The complexity of the solution will of course also have an impact on this, which is probably the reason that not many respondents bring it up. This is further supported by: "(…) it's simple, but it's not (…) once you get into sort of the details of (…) things that you can choose within the flow I think are a little bit hard to sort of find and understand" (Vicki).

As explained previously, the solutions are improved with new features rather informally based on user feedback: "during testing we've already got some feedback on potential improvements" (Vicki). This is a general pattern followed in all the cases that launches a version of the solution to the users as part of their testing. Change management seems to be handled in a similar manner across the use cases as all of them are launching the solution to their team or a smaller part of it and taking their feedback in consideration for the improvement of the solution (Vicki; BMC; InfoSec 1; Vishal). None of them have an actual process defined for change management and since virtually no documentation exist on the solutions, these changes are also not documented. However, as BMC puts it: "We can't account for everybody's feedback (…)" (BMC), which is the case across all of the cases. It seems like the feedback is for the most part collected and then the decision to accommodate that is made at a

later point by the developer or the entire team. This is the case with the AsiaPac quality team: "(…) as we get sort of more feedback and think oh this could be improved. We then basically create a version two (…)" (Vicki). The feedback collected in this case is from the end-users which are not the team themselves but engagement managers. In this case the feedback act more as suggestions as the end users cannot all be gathered in the same room to discuss an optimal solution. In the case of BMC where the end-user are the same team as is developing it, they do it differently: "(…) major like updates were you know obviously consulted by was obviously consulting with, I guess multiple stakeholders and whether you know that feedback was something that they all needed" (BMC). This is an approach also followed by other respondents as the end users are easy to gather for further discussions (InfoSec 1; Vishal).

The use of the Power Platform to optimize and automate own or team work is based on experienced pain points. Close to no documentation exist in all stages of development and the design considerations are made concurrently with development based on various factors. Close collaboration with the users is possible as the developer usually is part of the end-user group, leading to continuous improvements on the solution. Maintenance of solutions are kept with the developer or his team, however little concern is given to this as the simplicity of the solutions lead to seldom breakdown.

Summarizing the three scenarios

The three scenarios have very different ideas on how to best use the Power Platform. One Scenario follow a very professional approach in delivering client solutions and they are thus named hereafter. The second scenario begins by following a structured approach, but abandon throughout a project in favor of getting the job done like a Taskforce on a mission. The third scenario follow no process at all and basically build whatever they want, however they want, resembling the scenario of a cowboy in the wild west. The goals and approaches of the three use cases are summarized below:

	External formal use "The Professionals"	Internal formal use "The Taskforce"	Internal informal use "The Cowboys"
Goal	Building solutions for external clients increasing sales	Digitizing and optimizing internal clients work reducing operational cost	Automating or optimizing own or team's work reducing inconvenience
Specification	Formally gathered requirements specified in a formal agreement getting client sign-off prior to development.	Formally gathered requirements specified in a requirements document requiring no official sign-off.	Informally gathered requirements based on team meetings or own knowledge, with no specification of these.
Development	Work in sprints delivering increments to client and utilize first sprint to understand requirements.	Structured approach to work following either sprints or a plan and adapts to change throughout development.	No formal structure of the work, solutions is a constant work in progress implement changes ad-hoc.
Validation	Structured approach to testing utilizing three types of testing with user acceptance test marking the official handover to the client.	Seemingly structured approach to testing onboarding small user groups at the end of development prior to full-scale.	Unstructured approach to testing utilizing the concept of trial and error in development efforts and early launch to teams when first version is ready.
Evolution	Time limited maintenance formally agreed with client, addition of new functionality when agreed with client.	Maintenance kept with developer and team, taking on feedback and planning implementations based on feedback.	Maintenance kept with developer and team having little concern as stability is high and new functionality is added ad-hoc.

The Professionals

The Professionals are so named as their approach overall is very structured and successfully follow best practices in software engineering. Solutions are broken down and delivered in increments suggesting incremental development (Sommerville, 2015). These increments are identified in advance and delivered through iterations within the project, which is a not so uncommon mixture of the plan-driven and agile approach to incremental development (Sommerville, 2015). The reason they follow this mixed approach is that they need to manage scope and ensure alignment between EY and the client, so that a contract can be drafted and signed by the parties ensuring payment for EY and delivery to the client.

The Professionals follow the requirement engineering process in gathering requirements of the solution (Sommerville, 2015). Requirements elicitation is done through an initial analysis phase including discussions and interviews with the client. This results in specification documents such as process flows and initial mock-up as well as the high-level requirements or user requirements of the solution. These high-level user requirements are usually specified as natural language user requirements containing both functional requirements in terms of i.e. specific features of the solution and non-functional requirements in terms of i.e. volume of data or rules surrounding the solution (Sommerville, 2015). These high-level requirements can also be related to userstories as they too are quite abstract in their definition of the software. Following the specification step the requirements are reviewed with the client and the client sign-off on the requirements (Sommerville, 2015).
After the client sign-off is received on the initial requirements the Professionals start transitioning into a more agile approach by working in sprints, which is a term that comes from the agile methodology "Scrum" and refers to an iteration of development (Sommerville, 2015). The first sprint is dedicated to further specifying the details of the solution in terms of technical infrastructure, resources needed and other details. This is possible in the scrum methodology as a product backlog can contain items to produce design and architectural documents as well. This dedication of an entire sprint can however also be related to the spiral nature of the requirements engineering process, where the outer ring of this spiral contains "System requirements specification and modeling" (Sommerville, 2015:112). By using design sprints as a transition from plan-driven requirements engineering to agile development, the Professionals are allowed to seamlessly go into development with an agile mindset.

During the development phase the Professionals continue working in sprints after which they will release a product increment to the client. This is in line with the scrum methodology which states that after a sprint a potentially shippable software is ready (Sommerville, 2015). After each release client feedback is gathered allowing the team to adjust their development efforts according to client needs, which is also one of the known benefits with incremental development (Sommerville, 2015). In their delivery of these product increments they adopt the concept of the Minimum Viable Product. This

concept is formally defined as: "(…) that version of the product that enables s full turn of the Build-Measure-Learn loop with a minimum amount of effort and the least amount of development time" (Ries, 2011, p.77). An MVP is synonymous with the early increments of incremental deliveries: "(…) the early increments of the system include the most important or most urgently required functionality" (Sommerville, 2015:50). Thus the use of MVP's in their deliveries aligns well with the concepts of incremental deliveries. The benefit of them adhering to this principle is that it allows them to build a scraped, but working product, measure the clients response and learn from the feedback before delivering the next increment - i.e. a full circle of the build-measure-learn loop (Ries, 2011). Sommerville (2015) state that the product increment not necessarily have to be deployed with the customer, but it is enough for them to be exposed to the increment, which is the approach that the Professionals take. This approach could be due to the waterfall methodology overarching the process as a contract is signed in the beginning and a product is delivered in the end resulting in a payment transaction from the client to EY.

Incorporated in these sprints the Professionals embed some testing activities. First of all the teams have the developer test the individual pieces of software prior to releasing it to in the current increment. This type of testing is known as unit testing (Sommerville, 2015). Each of the components being developed by a team member is tested by that same team member. This type of testing should not be confused with component testing which is focusing on the interaction of multiple components (Sommerville, 2015). Besides this, testing is done by allowing a group of users access to a pre-prod environment or doing some integration testing. The first type of testing is user testing and more specifically beta testing (Sommerville, 2015). The second type of testing is another development testing approach testing the entire system with all of its integrated components, also known as system testing (Sommerville, 2015). The focus of this test is on the interactions between the components and an effective way of conducting such a test is through use case-based testing, as the use case force these interactions to occur (Sommerville, 2015). Use case based testing align well with user stories, which is the agile approach to capture client requirements. The Professionals will always end their delivery with a user acceptance test, which essentially entails having the user test out the entire system and accept it as it is or require further development on it (Sommerville, 2015). This type of testing is often used in custom systems development, which is the exact type of cases that the Professionals deal with. This testing activity mark the end of the delivery, tying it back to the high level user requirements defined by the client, and the system is handed over to the client. Agile methodology does not prescribe any one type of testing, however automated testing is suggested in Extreme programming (Sommerville, 2015). This is something the Professionals sometimes do when dealing with high testing volumes, however this is not part of their standard delivery model. The testing activities - Unit testing, systems testing and beta testing - although stemming from plan-driven software development, work well under agile ways of working. One problem that could arise from the

systems testing approach, is that the tester ends up being a bottleneck preventing presumably finished functionality from being included in the current increment. The user acceptance testing is considered redundant in agile methodology as the user is closely involved in the development process (Sommerville, 2015). This is where the Professionals transition back into the plan-driven approach, as they need to tie up the project and hand it over to the client ensuring the client is satisfied with the result. The utilization of traditional testing approaches mixed with agile deliveries is not uncommon as it is dificult to involve the customer to the extent agile methodologies suggest (Sommerville, 2015). This is further enhanced for the Professionals as ther clients are external and reside in other organizations.

Having handed over the solution to the client, the Professionals would not concern themselves with the further evolution of the solution. The client can sign a maintenance contract with the Professionals, however this would usually be for a limited period of time. Maintenance includes fault repairs, environmental adaptation and functionality addition (Sommerville, 2015). In general the maintenance contract a client can sign is pertained to the support of the system, however for a limited period of time as the Professionals are not interested in long term maintenance. It is very common that custom software is taken over and maintained by the client themselves or an external party takes over the maintenance and further evolution of the system (Sommerville, 2015). As the Professionals employ a hybrid of plan-driven and agile approaches, they avoid the challenge of not having proper documentation in place when handing over to the evolution team with the client. However as the Professionals for the most part, do not develop automated tests, such a challenge might still occur in the case that the evolution team wish to follow an agile approach (Sommerville, 2015). With regards to the other two maintenance elements, environmental adaptation and functionality addition, the Professionals being interested in new projects also offer the client to come back with new requirements in the future. Should the client want to add new functionality in the future, the entire cycle from requirements taking to client handover is repeated.

The Taskforce

The Taskforce have gotten their name as their approach is structured but focused more on results rather than the process. They also adhere to a hybrid of agile and plan-driven development, however a less optimized hybrid seems to be used. Most of the requirements are identified up front, however further requirements are taken throughout the project. Best practices are considered but not a priority throughout the project.

When it comes to requirements gathering, the Taskforce follow a similar, but slightly different process to the Professionals. Meetings are held with the end-users of the solution to discover what

current state is and what their pain-points are and these initial meetings result in some high-level user requirements as with the Professionals. However rather than following the process of requirements engineering or agile, these teams follow more of a "Service oriented engineering" approach. This approach entails the use of existing cloud based services to: "compose and configure services to create new, composite services" (Sommerville, 2015:541). The Taskforce first attempt to gain an understanding of the workflow, after which they would research for what is possible either by reading documentation or try and build a POC. A POC is the same as a prototype, as it allows for the users or developers to get an early understanding of limitations and architectural design decisions can easier be (Sommerville, 2015). The process of defining workflows and research possibilities, resemble the first two steps in service oriented software engineering - formulate outline workflow and discover services (Sommerville, 2015). Based on this initial research or exploration, they either define a solution architecture or refine existing documentation. This resembles the next two steps in service-oriented software engineering - select services and refine workflow (Sommerville, 2015). This process is different from the Professionals, as the Professionals do not consider what is possible in their requirements taking as their primary goal is satisfying the client and custom solutions result in more hours billable. The Taskforce on the other hand would try and push back on requirements based on what is possible, which could be due to the client and developers are governed by the same organization. Being part of the same organization establish a common ground for understanding the barriers that the governance creates and the time it takes to overcome these. The Taskforce does not have an official contract in place and thus no-official sign-off is required as is the case with the Professionals. The process for the taskforce process although seem sequential actually follow a more iterative nature as constant changes are introduced throughout development, which would then initiate further research and refinement.

Following the defined workflow above the Taskforce would start converting process steps into an executable program, which might involve web-based user interface to interact with the workflow logic (Sommerville, 2015). In this process step the Taskforce would commit themselves to predominantly one software engineering approach i.e. agile or plan-driven. This means that they might work in sprints developing based on user stories or they might follow a more sequential approach developing based on requirements. (Sommerville, 2015). Regardless of approach the Taskforce will keep the dialogue going with the users throughout development. Depending on what methodology is followed this dialogue would either involve the developer or be controlled by the project manager as a middleman (Sommerville, 2015). The Taskforce adapts to change throughout the development of the solution making corrections based on user feedback, which is something they borrow from incremental development (Sommerville, 2015). These changes are either documented as formal requirements or simply take the form of feedback based on product demonstrations depending on what methodology is followed. Solution documentation throughout development is not a priority

for the Taskforce, as the emphasis is put on delivering a working solution to the users. The solution is delivered to the user after all development efforts have occurred, which resembles the plan-driven approach that the Professionals were also adhering to. They might expose the user to an unfinished version as for instance with the POC as mentioned previously, however this is not a priority in the development phase.

During the development phase mentioned above the Taskforce would conduct development testing in the form of unit testing as was done with the Professionals as well. The testing activities could also be considered component testing depending on what constitutes a unit (Sommerville, 2015). A service in itself could be considered a unit and in that case connecting multiple units would make a component and thus the developer testing employed would be component testing. On the other hand a unit could also make up individual features of the solution connecting multiple services, in which case unit testing is the applied testing technique. In addition to this, system testing would often be applied, where the entire system is tested. As explained with the Professionals this type of testing is most effective following use-case based testing. The Taskforce utilize just that type of testing by having a tester or the developer themselves test different use cases in the system. A final type of testing the Taskforce make use of is beta testing, where a couple of users are allowed early access to a release (Sommerville, 2015). Just like with the Professionals user testing is applied in the end of a project to mark the end of a delivery. However where the Professionals would apply user acceptance testing ensuring client sign-off, the taskforce would apply beta testing as no official sign-off is needed and the solution is not handed over to the client. This testing activity reveals only minor changes as the Taskforce has adapted to change throughout the entire development phase.

After having received initial feedback based on the beta testing, the Taskforce would deploy the solution on full scale. After this deployment, which is happening in the production environment of EY, the Taskforce will continue to maintain the solution. In the initial period of deployment they will be more available to maintenance as early bugs or changes are expected to arise and these are prioritized to be fixed instantly. Following the hypercare period, the Taskforce continue to monitor and maintain the solution including the fixing of bugs, environmental adaptation and addition of new functionality (Sommerville, 2015). For the addition of new functionality the Taskforce will take in requirements or feedback and plan for the implementation of the changes when it suits their schedule. Some changes are also implemented on the Taskforce's own initiative to improve performance or reduce complexity of existing components in the solution. This type of improvement resemble the principles in agile methodologies in terms of continuous improvements to the solution. As the Taskforce, whenever they see opportunity for improvement, go in and refactor the components to reflect a better more simplistic service (Sommerville, 2015).

The Cowboys

The Cowboys are so named as their approach to development is following pretty much whatever methods they feel like. Almost no considerations are made in terms of what best practice is when developing solutions. As such, no specific software engineering approach is followed, however some principles from the agile mindset are utilized although this not being a conscious decision. The four elements of software engineering, specification, development, validation and finally evolution are all overlapping in the Cowboys approach.

As the Cowboys are developing what they want for either themselves or their team, requirements are either fully or partially known to the developer prior to project initiation. In the case of team based solutions requirements are taken through informal conversations with their team and presumably noted down somewhere. This is done continuously throughout the entire software engineering process, which resembles the agile principle of: "Customer collaboration over contract negotiation" (Sommerville, 2015:76). The Cowboys would go to the Power Platform and start investigating what is possible, just like the Taskforce would do, however unlike the Taskforce, no official design or development documents will be drafted (Sommerville, 2015). Instead the Cowboys will design the solution in a notebook or even directly in executable code, which is in line with the agile manifesto: "Working software over comprehensive documentation" (Sommerville, 2015:76).

Throughout development the cowboy will follow a code and fix way of working, where they would just start developing a solution and in case any errors come up, they will find a work around. This approach once again aligns with the agile principles, specifically the: "responding to change over following a plan" (Sommerville, 2015:76). The architecture of the solution is very much influenced by available services within the enterprise as they often have no mandate to request a new service being acquired. Sometimes the Cowboys will utilize existing solution templates and configure them or get inspiration from them when designing the architecture of the solution. This constitutes reuse on the system level or abstracted level respectively, minimizing the efforts needed in designing solutions (Sommerville, 2015). Just like the Taskforce, the Cowboys would often have the user experience in mind throughout the development entailing a responsive user interface and one stop shopping type of thinking. This could be due to the fact that they themselves often would be the users and thus they know exactly how to best optimize their work. As they also constitute the users or their team does, the constant involvement of themselves and teammates in the development process aligns well with the principles of agility (Sommerville, 2015). Close to no documentation is produced about the solution, its functionality or reasoning. The Cowboys often develop in the production environment, however they are aware of the challenges with developing on production data and most of them attempt to account for this in the development and testing stages.

The testing activities are also very unstructured, however they do conduct some development testing in developing the components of the solution. In their approach to development testing they follow the same approach as both the Professionals and the taskforce, testing individual units or components of the solution. In doing so they would ensure the data underlying the unit or component is amended for testing purposes preventing the interaction with production data. They do however not utilize any structured approach to this testing activity, such as thoughtfully identify and test individual units based on a guideline or partition. This unstructured approach induce increased risk of bugs occurring in the future. Besides the development testing, the Cowboys also apply user testing, although the approaches to user testing varies. As some of the Cowboys also are users themselves, they could be seen as alpha testers closely involved in testing the solution from the user perspective (Sommerville, 2015). However the developer taking upon the role as a user could also be interpreted as systems testing as explained with the Taskforce. In the cases that the solution is targeted users other than the developer, beta testing is utilized and a small group of teammates are asked to test out the solution (Sommerville, 2015). The development of the solution follow a less structured but iterative nature, suggesting the use of incremental development and once again aligning with the core agile principles.

When the cowboy have a working version of the solution ready, it is released to the users. From that point and forward the cowboy commits itself to maintain and further improve on the solution based on user feedback, whether that be teammates or users outside the team. Cowboys value simplicity in their solution designs and as such very few of them actually experience failures once it is live. When they do experience errors, the developer will go in and fix the bug, sometimes refactoring the solution to prevent future bugs of that kind. This approach is commonly known as an extreme programming technique within the agile domain (Sommerville, 2015). Cowboys are often challenged by troubleshooting as the services used in the solution are abstracted and the source code is maintained with Microsoft. This challenge is very common when following a service-oriented development approach and was also experienced by the Professionals (Sommerville, 2015). However, the Professionals have an advantage in these situations as they have a contact in Microsoft helping them out with debugging from time to time, a luxury the cowboy does not have. Cowboys seem to value Google higher than both the Professionals and the Taskforce and use it frequently for development as well as debugging. Adding new functionality is by the Cowboy done based on user feedback on an ad-hoc basis when they have time for it. Sometimes the feedback is structured and analyzed together with the team other times it is just implemented. The Cowboys are able to follow an agile approach in software evolution as they are closely collaborating with the users everyday and based on that can continuously improve the solution (Sommerville, 2015).

The Power Platform is being used in three different scenarios labelled as the Professionals, the Taskforce and the Cowboys. These labels are based on their approach in developing solutions with the

Power Platform, ranging from structured to unstructured. The Professionals follow a combination of agile and plan-driven software engineering in their use of the Power Platform resulting in thorough documentation, high-quality assurance, no operational cost and potential for increased sales of solutions. The Taskforce attempt to follow best practices, but in the end delivering results takes precedence. This approach results in adequate documentation of the software, medium-quality assurance of applications, increased operational cost and potential reduction of overall cost. The Cowboys deviate from best practices based on convenience, leading to their approach following an entirely agile mindset. This approach results in no documentation of software, low-quality assurance of applications, small increase in operational cost and potential reduction of overall cost.

EY's governance of the Power Platform

Having discovered and analyzed the different use case scenarios, this section will analyze how EY is addressing the governance of the Power Platform, fostering desired IT behaviors. In doing so the applied governance will be analyzed using the governance mechanisms described by De Haes & Van Grembergen (2004). This will be followed by an analysis of the how IT governance decisions are made using the framework by Weil & Ross (2004) and finally an analysis of the IT organization through the concept of paradoxes and bimodal IT by Jöhnk et al. (2019).

Applied governance

The organizational structures surrounding the Power Platform is an applied governance mechanism to ensure IT and business alignment as suggested by De Haes and Grembergen (2004). A product manager is within the end-user technology charged with the duty of: “(…) to try and, you know, get our [EY's] money back on the investment (…) my success be measured by how much I increase adoption (…)” (Eoin). Although placed within the technology organization of EY, his interest are bound in standard business interests such as return on investment of the Power Platform. The management of the platform is further under the influence of some key stakeholders to the product manager in terms of information security and client technology’s low-code division. The information security team's responsibilities are to "review and assess risk on platforms that are specific to internal end user EY people” (Security Consultant). The low-code division in client technology does not have an actual stake in the use of the platform internally, however as the product manager state that they work closely together, they serve as an influencing party. Their responsibility is “(…) building consumer grade, reusable, revenue generating applications that are (…) used by EY employees but also used by our clients” (Anbu). While the product manager is responsible for increasing adoption and getting a good ROI, the information security team and low-code division are more focused on IT goals such as security, IT Risk, Reusability and development of IT. With regards to the Power Platform all seem to have a clear understanding of what their roles and responsibilities are, which is

key element in defining proper structures of business and IT alignment (De Haes & Van Grembergen, 2004).

In addition to defining proper structures of the IT governance, proper processes should also be introduced in the IT organization (De Haes & Van Grembergen, 2004). To prevent people from taking internal information and accidentally post it on social media, general environment strategies are applied in combination with a Power Platform function called Data Loss Prevention policies. This strategy essentially involves splitting the tenant into different environments: "we have the default environment which we currently use as personal automation (…) the non-default environments (…) to a more enterprise type solution building (…) then we have the dedicated environments which are used for like those niche (…) solutions" (Security consultant). This approach is then combined with a tailored DLP for each environment, such as the default environment where "pretty much people can just do stuff with Office 365 connectors" (Eoin). This strategy is applied to prevent people from intentionally or accidentally to leak information: "We don't want people within EY taking a document from that point and posting it to Twitter or LinkedIn (…)" (Anbu). On one hand this help the IT department fulfill their responsibilities and keep the organization secure. On the other hand, the restrictions are restricting the business in what they can use the platform fore and thus risk Impeding on the adoption on the platform. To accommodate situations where the business needs additional functionality than what is in the default environment, a process has been introduced for the business users to get access. This process includes IT approval as the business have to: "(…) engage information security and you know ensure they're happy with whatever you're doing" (Eoin). This will allow the business to fulfill their needs of utilizing the platform in complex use cases, while simoultaneously account for IT interests of security and risk.

Processes are also introduced to deal with the new features on the platform as these are introduced by Microsoft ongoingly. The product manager terms the features of Microsoft as "half-baked" as they are great capabilities from a business perspective, but they lack focus on governance functionalities (Eoin 2). Both information security and the low-code division echo that "Microsoft is playing catch up on the security side(…)" (Security Consultant) and "(…) governance will become a big topic and I think Microsoft will address it too (…)" (Anbu). In coping with this challenge two processes are introduced handling small and big changes. The product manager explains that:

> "I'm not a security specialist, I'm not a data privacy specialist, but I've got a view on. Information security and I've got a view on data privacy (…) the features that are being rolled out in general month by month, none of them require, you know, a data privacy review (…) none require infosec" (Eoin 2)

As the product manager states he has an idea of how these changes will impact data privacy and information security, which enables him to handle small changes throughout the year. However when

Microsoft announces bigger features at their "Ignite conferences", these require the involvement of information security as their impact is much bigger. The product manager explains:

> "(…) the big Ignite announcements, like their much bigger work you know, like Microsoft released that data verse for teams product back in September (…) great capability that they release really interesting gets us away from you know, this legacy SharePoint list as being a data storage. But you know (…) there was, you know, no governance around the whatsoever when released. This is improving." (Eoin 2).

This demonstrates the challenge of balancing IT interests with business interests as new capabilities allow the business to innovate and evolve, while in the same time compromising IT governance concerns. The product manager states how the announcement of these capabilities further challenge him in his budget planning: "(…) but then Microsoft can come along and throw teams for power apps at me and it's like you know it screws up my budget for the year" (Eoin 2).

Further than specifying structures and processes pertained to how this platform should be governed, if business and IT do not understand each other they will have little impact (De Haes & Van Grembergen, 2004). Relational mechanisms are defined to foster this mutual understanding between IT and Business. With relation to the Power Platform this understanding is fostered by providing policies and guidelines in how to use the platform. This is done through a central Power Platform SharePoint site where people can find "Protection guidelines and (…) best practices" (Eoin 2). The security consultant also mention policies as a way of governing the use of the Power Platform: "we have acceptable use policies (…) there's the code of connection policy" (Security consultant). These policies provide the Power Platform users with the guidelines needed for developing in the Power Platform and help them understand why IT interests are important to consider. The management defined business strategy also serve as a mediator, however this usually focus on getting IT to understand the business interests.

The collaboration between information security, low-code division and the product manager also serve as a relational mechanism further aligning IT and business. This alignment is further supported as the product manager states that: "It's not a case of let's stop it. It's a case of figuring out how we use it safely and security. Generally, that's the way the conversations going, in EY" (Eoin). This suggest that IT understands the need for the tool as well as the business understand the need for security. This is further supported by both the IT entities who acknowledge low-code development tools as the primary way of delivering applications in the future (Security consultant; Anbu). This understanding of business interests sometimes lead to the enablement of features on the platform that compromise the interests of IT. The product manager mentions the data verse for teams as a concern multiple times: "I know there's a whole bunch of things in there that let people work around our, you

know, governance process in place (…), I've concerns from that point of view" (Eoin 2). The fact that this capability is enabled in spite of options to circumvent existing processes supports the argument of IT understanding the business. Furthermore, the concerns of the product manager supports the argument of business understanding IT.

Both business and IT agree that the governance mechanisms put in place need to be balanced against the autonomy of the employees to use this platform.
The product manager states: "The balance between innovation and locking a system down with so much bureaucracy and processes where people are just bored like that where we are much more in the innovation side" (Eoin 1). This is supported by the low-code division lead: "I don't think you know we can limit our business users from creating applications. The challenge is like on what is the right level of check and balance (…)" (Anbu). Both statements testify to the balancing of on the one side having governance mechanisms in place to ensure correct and efficient use of the tool and on the other side allowing users the autonomy to build solutions themselves. This balance can be related to the discussion on centralization versus decentralization in IT governance (Weil & Ross, 2004)

Decentralization of IT decision making

The way in which EY is balancing autonomy with control on the platform is best illustrated through the decision-making archetypes and key issues of IT governance (Weil & Ross, 2004).

Business application needs entail the "business requirements for purchased or internally developed IT applications" (Weil & Ross, 2004:4). Decisions related to what applications are needed in the market is in the case of the Professionals made by themselves based on client needs. For the Taskforce and the Cowboys these decisions are made internally by either a small group acting as clients to the taskforce or themselves based on their own needs. This can be sided with having an anarchist governance structure in assessing the business application needs (Weil & Ross, 2004).

Prioritization and investment decision entail: "decisions about how much and where to invest in IT, including project approval and justification techniques" (Weil & Ross, 2004:4). In all three cases the teams themselves are making the decisions for whether or not they should develop a certain IT application. This once again points to the anarchist decision making archetype (Weil & Ross, 2004)

The decentralized nature of Business application needs and IT investment and prioritization should be considered in light of the previous structure. The product manager states that:

> "(…) enabling the business to solve their own problems (…) it's much easier than you know, going through a very long process of hey IT, I want to do something OK, you know, let's get

> that budget approved, then let's get a business analyst over and let's get a developer on (…)" (Eoin 1)

This suggest that decisions regarding both business application needs and IT investment and prioritization was much more centralized prior to the introduction of the Power Platform.

Although the IT governance on Business application needs and IT prioritization and investment is decentralized to the extreme, some of the decisions concerning enterprise wide usage of the platform is governed at a higher level. IT infrastructure entails the: "centrally, coordinated, shared IT services providing the foundation for the enterprise's IT capability and typically created before precise usage needs are known" (Weil & Ross, 2004:3). The platform in itself is providing a shared IT capability to the business as was demonstrated through the three use cases. Decisions to implement this platform and how much to enable is pertained with the product manager of the platform and his stakeholders. As described previously through the structures, processes and relational mechanisms, all three parties seem to have decided that this platform constitutes a valuable contribution to the IT infrastructure of EY. As the decision to implement this platform in the entire enterprise was made involving both business and IT, the decision making archetype resemble that of IT duopoly (Weil & Ross, 2004).

Decisions related to IT architecture related to the Power Platform seem to follow the same archetype. IT architecture entails "an integrated set of technical choices to guide the organization in satisfying business needs" (Weil & Ross, 2004:3). With regards to the Power Platform, IT architecture can be related to how the platform is technically restricted to guide the appropriate use of the platform. As illustrated through the process governance mechanisms these decisions are also taken by the product manager in collaboration with his stakeholders once again pointing towards the IT duopoly decision making archetype (Weil & Ross, 2004).

The general role of IT in the business in terms of IT principles, are presumably decided higher up in the ranks, however from the dataset it is not possible to tell. Mapping these decisions and decision-making archetype in a table paint a decentralized picture of EY's governance:

Archetypes\Issues	IT principles	IT architecture	IT infrastructure	Business application needs	IT investment and prioritization
Business monarchy					

IT monarchy					
Federal					
IT Duopoly		X	X		
Feudal					
Anarchy				X	X
Don't know	X				

Weil & Ross (2004) further connects the IT governance to business goals and find that decentralized governance models are often found in businesses performing on growth, which aligns very well with the goals of the product manager (Eoin 1). The decentralized IT governance set-up prioritize local innovation with communities of practice rather than low business cost through standardized business processes (Weil & Ross, 2004). The product manager seem to try and accommodate both:

> "At the end of the day you do want to absolutely drive innovation, but you want to ensure people are, you know, following the right steps, things are getting documented and you know EY processes are being followed" (Eoin 1).

Although the governance structure seem very decentralized, some focus is still on controlling the use on the platform.

Decentralized bimodal IT

This conflict of the IT organization being both innovative and efficient is a paradox which has been addressed through the nature of bimodal IT (Jöhnk et al., 2019). As explained earlier bimodal IT means that the IT organization have two modes – Traditional IT and agile IT. Bimodal IT is known to take one of four Archetypes, however in the case of EY and the Power Platform another archetype appear.

Agile IT is focusing on flexibility, innovativeness and experimentation, time-to-market, and customer needs (Jöhnk et al., 2019). As was demonstrated through the governance in one page framework, application development and decision making is heavily decentralized. This results in IT being developed all over the organization and as the use cases show, these developments utilize agile principles in their development efforts. Meanwhile as demonstrated through the analysis of the governance mechanisms related to the platform, a centralized IT function attempts to balance the risks and efficiency of the platform. This resemble the traditional IT department striving for stability, reliability, efficiency, and long-term planning (Jöhnk et al., 2019). This comprise a whole new

archetype as the business on the lowest levels are constantly innovating and building applications using the platform and the IT departments are increasingly concerned with security and efficiency.

This archetype specifically highlights the paradox of integration vs. autonomy as found to be an antecedents of the ambidexterity paradox (Jöhnk et al., 2019). The decentralization of Agile IT allows for close to total autonomy as the use cases in this research illustrate. Simoultaneously restricting mechanisms are introduced in an attempt to align agile IT with the security standards of traditional IT as illustrated in the analysis of governance mechanisms.

Furthermore this new archetype highlights the paradox of business/IT versus IT/IT alignment. This is first of all due to the business being enabled in development with no IT experience the business and agile IT are brought closer together. However this is at the expense of traditional IT and agile IT alignment. First, reusability is reduced and by extension efficient use of resources heavily reduced: "(…) if someone has actually built something that is going to be useful for others, there's no easy way (…) to share it with others to consume" (Anbu). Second security standards are compromised: "(…) am I confident that everybody in EY is compliant in how these tools are used? I am aware of gaps." (Eoin 1).

Discussion

This paper sought to understand how the use of low-code development platforms impact organizational interests. Through an inductive case study of how the Microsoft Power Platform is being used and governed within EY, this paper provides two significant findings.

Proper use of the Power Platform

The first finding suggest that the Power Platform is being used in three different scenarios, which can be mapped across two dimensions – organizational scope and formality of agreement. Through a thorough analysis of the software engineering practices in each scenario it was discovered that the three approaches stretched from structured to unstructured software engineering practices leading to the classifications as the "Professionals", "Taskforce" and "Cowboys". The two-dimensional plot is displayed below:

Formality of agreement

Organizational scope	Formal	Informal
Internal	Taskforce	Cowboys
External	Professionals	Speculative

The two-dimensional nature of the use cases suggest the existence of a fourth use case in terms of "external informal use". As projects reaching across organizational boundaries often require some control in terms of contracts to scope what is being developed, one can question whether such a use case exists. This raises questions as whether the dimensions of this model is representative to the identification of use cases. One could argue that the use cases move along a continuum from structured to unstructured software engineering practice. Although this continuum is accurately describing the approaches of these use cases, it provides only a partial picture as factors influencing the software engineering approach is not divulged. Furthermore, the practices of software engineering in all cases are expected to improve based on experience thus leaving the continuum approach rather useless. The author would rather argue that the fourth use case exists. This use case would be highly dependent on trust and close collaboration, as no contract or formal agreement on requirements are present to govern the relationship. In a consulting practice this might be done to nurture client relations or win over a client by demonstration of capability or perhaps even a charity case for a non-profit organization. Such a case was not found in this study, leaving a gap for further research.

Decentralized bimodal IT

The second finding suggest that the introduction of the Power Platform enterprise wide lead to a new archetype of bimodal IT organizations – decentralized agile IT. Through analysis of the governance structures and mechanisms surrounding the platform it was discovered that IT decision making was increasingly decentralized leading to rise Agile IT throughout the entire organization. The introduction of such a bimodal organization nurtured the existing paradox of ambidexterity, while simultaneously giving rise to new ones previously found by Jöhnk et al. (2019) in terms of alignment as well as collaboration.

Whether or not the decentralized set-up allowing business users to develop applications constitutes an actual IT organization is questionable. Business users have always used business tools to develop small business applications such as extensive excel spreadsheets or Microsoft access databases. However what differs with the Microsoft Power Platform is that it allows for more elaborate applications such as integration with other applications as well as some whole new opportunities for scaling. Thus the decentralization of such capability constitutes the decentralization of IT capabilities and by extension the IT department.

Furthermore, this new decentralized IT organization was suggested to be a whole new archetype of bimodal IT. This can also be argued as archetypes does not consider the degree to which an IT department is centralized as an factor in the different archetypes. Ignoring the decentralized nature of the IT department, Archetype A could also be used in explaining the structure of the IT organization. However, this archetype does not quite capture the dispersed nature of the organization. Although the use case scenarios identified support the use of different methodologies per projects, reality shows that increasingly agile methods are applied in the use of this platform. This suggest the separation of agile and traditional IT which lends support for archetype B or C depending on the interpretation of the divide between the two modes in the organization. However this paper finds that the extreme decentralization of agile IT result in a conceptual merge of business and agile IT, which is a nuance not captured by existing archetypes. Thus this paper finds a whole new archetype of bimodal IT with the introduction of the Power Platform.

Limitations

The data foundation of this research could additionally be questioned as only two use cases support the findings of the Professionals and taskforce. It is entirely possible that these two use cases are not representative of the population and thus conclusions made on best practices are not thoroughly supported. Furthermore, the external use cases are less detailed in their description of their practice as client secrecy prevented them from divulging too much. This challenges the generalizability of the findings pertaining to the Taskforce and especially the Professionals. This is however of less importance as the case-study strategy was adopted and thus focus is not put on generalizable results (Saunders et al., 2007).

In the analysis of the governance of the platform findings suggest a close alignment of business and IT as the product manager is treated as a business representative. Whether his role is residing within the roam of what constitutes business can be questioned as his role resides within the EY technology suggesting a more IT related role. As he mentions focusing on getting return on investment and driving innovation it was deducted that his role was a business role rather than an IT role suggesting close alignment of IT and business. Should his role be deemed to be within the roam of IT the

findings would be impacted in terms of decision making archetypes changing from IT duopoly to IT monopoly as well as a more unclear picture of business and IT alignment (De Haes & Van Grembergen, 2004; Weil & Ross, 2004).

Another point made throughout this that unit testing is utilized in quality assuring the application in all cases. However in theory unit testing involves a much more structured approach than what was found in practice for the taskforce and the Cowboys (Sommerville, 2015). As such findings built upon the fact that units are thoroughly tested would be less valid.

Conclusion

This paper investigated the research question: *How does the use of low-code development platforms to build business applications impact organizational interests?* This was investigated through an inductive case study of EY's use and governance of the low-code development platform Microsoft Power Platform.

The use of this platform was found to follow three approaches to software engineering across two dimensions – organizational scope and formality of agreement.

The first identified approach was labelled "Professionals" as their approach follow a successful combination of best practices in plan-driven and agile software engineering. This approach results in thorough documentation of the software, high-quality assurance of applications, no operational cost and potential for increased sales.

The second identified approach was labelled "taskforce" as their approach attempt to follow best practices, but in the end delivering results takes precedence. This approach results in adequate documentation of the software, medium-quality assurance of applications, increased operational cost and potential reduction of overall cost.

The third identified approach was labelled "Cowboys" as their approach deviate from best practices based on convenience, leading to their approach following an entirely agile mindset. This approach results in no documentation of software, low-quality assurance of applications, small increase in operational cost and potential reduction of overall cost.

The governance of the platform was found to follow a decentralized structure leading to the rise of agile IT throughout the entire organization.

The decentralized structure of IT governance in the organization foster overall growth and local innovation. The paradox of organizations seeking to balance innovation and efficiency was also found in this case creating tensions between Business and IT. Combined with the abilities available in the Power Platform this led to agile IT being enabled and distributed throughout the organization

introducing a new archetype of bimodal IT. The introduction of decentralized agile IT introduced new paradoxes in the organization in terms of business/IT versus IT/IT alignment and integration versus autonomy in collaboration.

In assessing the overall impact on organizational interests, innovation is increasingly prioritized at the cost of efficiency and security leading to increased risk of overhead cost and data breaches.

References

Alexander, F. (n.d.). *What Is Low-Code? [2021 Update]*. Outsystems.Com. https://www.outsystems.com/blog/posts/what-is-low-code/

Arora, R., Ghosh, N., & Mondal, T. (2020). Sagitec Software Studio (S3) – A Low Code Application Development Platform. *2020 International Conference on Industry 4.0 Technology (I4Tech)*, 13–17. https://lutpub.lut.fi/bitstream/handle/10024/158441/masters_thesis_virta_tatu.pdf?sequence=1

BANSLER, J. O. R. G. E. N. (1992). A. Friedman with D. Cornford, Computer Systems Development: History, Organization and Implementation, Wiley, 1989. ISBN 0–471-92399-0, £35 (hard back), £ 18·50 (paperback). *Behaviour & Information Technology*, *11*(4), 245–246. https://doi.org/10.1080/01449299208924343

Bexiga, M., Garbatov, S., & Seco, J. C. (2020). Closing the gap between designers and developers in a low code ecosystem. *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 5–15. https://doi.org/10.1145/3417990.3420195

Clark, L. (2019, November 18). Low-code maturity boosts efficiency and helps user acceptance. *ComputerWeekly.Com*. https://www.computerweekly.com/feature/Low-code-maturity-boosts-efficiency-and-helps-user-acceptance

De Haes, S., & Van Grembergen, W. (2004). IT Governance and Its Mechanisms. *INFORMATION SYSTEMS CONTROL JOURNAL*, *1*.

Eliashberg, J., & Michie, D. A. (1984). Multiple Business Goals Sets as Determinants of Marketing Channel Conflict: An Empirical Study. *Journal of Marketing Research*, *21*(1), 75–88. https://doi.org/10.1177/002224378402100108

EY. (n.d.-a). *Global office locations*. EY.Com. https://www.ey.com/en_ae/locations

EY. (n.d.-b). *What we do - Our Services*. EY.Com. https://www.ey.com/en_gl/what-we-do

EY. (n.d.-c). *Who we are - Builders of a better working world*. EY.Com. https://www.ey.com/en_gl/who-we-are

FIEGENBAUM, A., HART, S., & SCHENDEL, D. (1996). STRATEGIC REFERENCE POINT THEORY. *Strategic Management Journal*, *17*(3), 219–235.

Forrester. (2014). *New Development Platforms Emerge For Customer-Facing Applications*. Forrester.Com. https://www.forrester.com/report/New+Development+Platforms+Emerge+For+CustomerFacing+Applications/-/E-RES113411

Gartner. (n.d.). *Definition of Citizen Developer - Gartner Information Technology Glossary*. Gartner.Com. https://www.gartner.com/en/information-technology/glossary/citizen-developer#:%7E:text=A%20citizen%20developer%20is%20a%20user%20who%20creates,built%20with%20tools%20like%20Microsoft%20Excel%20and%20Access.

Goswami, M. (2021, January 6). *What Is Low-Code Development?* Forbes. https://www.forbes.com/sites/forbestechcouncil/2021/01/07/what-is-low-code-development/?sh=1469a932496d

Gross, E. (1969). The Definition of Organizational Goals. *The British Journal of Sociology*, *20*(3), 277. https://doi.org/10.2307/588953

Humphrey, W. S. (1989). The software engineering process: definition and scope. *ACM SIGSOFT Software Engineering Notes*, *14*(4), 82–83. https://doi.org/10.1145/75111.75122

Kaplan, R. S., & Norton, D. P. (1997). Balanced Scorecard. *Das Summa Summarum Des Management*, 137–148. https://doi.org/10.1007/978-3-8349-9320-5_12

Kautz, K., Madsen, S., & Nørbjerg, J. (2007). Persistent problems and practices in information systems development. *Information Systems Journal*, *17*(3), 217–239. https://doi.org/10.1111/j.1365-2575.2007.00222.x

Khorram, F., Mottu, J. M., & Sunyé, G. (2020). Challenges & opportunities in low-code testing. *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. Published. https://doi.org/10.1145/3417990.3420204

Kotlar, J., De Massis, A., Wright, M., & Frattini, F. (2018). Organizational Goals: Antecedents, Formation Processes and Implications for Firm Behavior and Performance. *International Journal of Management Reviews*, *20*, S3–S18. https://doi.org/10.1111/ijmr.12170

Martins, R., Caldeira, F., Sa, F., Abbasi, M., & Martins, P. (2020). An overview on how to develop a low-code application using OutSystems. *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, 395–401. https://doi.org/10.1109/icstcee49637.2020.9277404

Microsoft. (n.d.). *Business Application Platform | Microsoft Power Platform*. Microsoft.Com. https://powerplatform.microsoft.com/en-us/

Oltrogge, M., Derr, E., Stransky, C., Acar, Y., Fahl, S., Rossow, C., Pellegrino, G., Bugiel, S., & Backes, M. (2018). The Rise of the Citizen Developer: Assessing the Security Impact of Online App

Generators. *2018 IEEE Symposium on Security and Privacy (SP)*. Published. https://doi.org/10.1109/sp.2018.00005

Ploder, C., Bernsteiner, R., Schlögl, S., & Gschliesser, C. (2019). The Future Use of LowCode/NoCode Platforms by Knowledge Workers – An Acceptance Study. *Communications in Computer and Information Science*, 445–454. https://doi.org/10.1007/978-3-030-21451-7_38

Rashid, F. Y. (2021, April 28). *Gartner says low-code, RPA, and AI driving growth in 'hyperautomation.'* VentureBeat. https://venturebeat.com/2021/04/28/gartner-says-low-code-rpa-and-ai-driving-growth-in-hyperautomation/

Ries, E. (2011). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses* (Illustrated ed.). Currency.

Rymer, J. R., & Koplowitz, R. (2019, March). *The Forrester Wave<sup>TM*: Low-Code Development Platforms For AD&D Professionals, Q1 2019</i>. Forrester. https://wwwcdn.spanishpoint.ie/wp-content/uploads/2019/04/The-Forrester-Wave%E2%84%A2_-Low-Code-Development-Platforms-For-ADD-Professionals-Q1-2019.pdf

Sahay, A., Indamutsa, A., Di Ruscio, D., & Pierantonio, A. (2020). Supporting the understanding and comparison of low-code development platforms. *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. Published. https://doi.org/10.1109/seaa51224.2020.00036

Saunders, M., Lewis, P., & Thornhill, A. (2007). *Research Methods for Business Students*. Prentice Hall.

Sommerville, I. (2015). *Software Engineering, Global Edition* (10th ed.). Pearson.

Stagner, R. (1969). Corporate decision making: An empirical study. *Journal of Applied Psychology*, *53*(1, Pt.1), 1–13. https://doi.org/10.1037/h0026849

Thompson, A., Peteraf, M., Gamble, J., & Strickland, A. (2015). *Crafting & Executing Strategy: The Quest for Competitive Advantage: Concepts and Cases (Crafting & Executing Strategy: Text and Readings)* (20th ed.). McGraw-Hill Education.

Vincent, P., Iijima, K., Driver, M., Wong, J., & Natis, Y. (2019, August). *Magic Quadrant for Enterprise Low-Code Application Platforms*. Gartner. https://d1wqtxts1xzle7.cloudfront.net/63556609/Gartner-Low-Noo-Code-Report20200607-123018-jta9ml.pdf?1591562455=&response-content-disposition=inline%3B+filename%3DLicensed_for_Distribution_Magic_Quadrant.pdf&Expires=1621234191&Signature=OIcltsACm7nCvdtJ0jooW-

8g6~Uyumbg3ATCXDIJ0pQg0tWExSIhx7hZ5TFdv8gMvYOz327tLyZoD9MBsGgO8uxhN6szvGzwGQN9Ph60ctB6tHXZMiDkOVfl-2dE7TTAHUZZ2tTwnPWartYJKIoIBl2P009GPWvSy0FFhzXboRFSx1yOOHQXeCqSiLP~-BXkS7ZuR-f9VwIYqqNYyn6RnoLmFbRGVb8AJRdfcCbg1njRm8NdVMVXkGBq0BfQlmyLShQ28m3lwqJelxSIRd8rZRde-G232lKbWJXUVUKtYHUMdgRUQX6ZA32ClwDD1oxMvLu5Y~glFE3XRu7V2Fvjag__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

Virta, T. (2018). *RELATION OF LOW-CODE DEVELOPMENT TO STANDARD SOFTWARE DEVELOPMENT: CASE BIIT OY*. https://lutpub.lut.fi/bitstream/handle/10024/158441/masters_thesis_virta_tatu.pdf?sequence=1

Waszkowski, R. (2019). Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, *52*(10), 376–381. https://doi.org/10.1016/j.ifacol.2019.10.060

Weill, P., & Ross, J. W. (2004). IT Governance on One Page. *SSRN Electronic Journal*. Published. https://doi.org/10.2139/ssrn.664612

Woo, M. (2020). The Rise of No/Low Code Software Development—No Experience Needed? *Engineering*, *6*(9), 960–961. https://doi.org/10.1016/j.eng.2020.07.007

Zolotas, C., Chatzidimitriou, K. C., & Symeonidis, A. L. (2018). RESTsec: a low-code platform for generating secure by design enterprise services. *Enterprise Information Systems*, *12*(8–9), 1007–1033. https://doi.org/10.1080/17517575.2018.1462403

Appendices

Appendix 1 – Use case interview guide

<table>
<tr><td colspan="2">Topic introduction: I am investigating the use of low-code development platforms in businesses, with a special focus on the use of the Power Platform in EY. One of the areas I am investigating is the governance around such platforms, which is where you come in.

Before we start the session, I would like to ask you a couple of formality questions:
- Can I record this session?
- Do you wish to stay anonymous?
- Would you like to have quotes send for approval?
- Do you have any questions or concerns you would like me to consider?</td></tr>
<tr><td>Introduction</td><td>Can you tell me a little bit about yourself?
• Name
• Age
• Title
• Previous work
• Education

What is your role at EY?
• Title
• Global/Area/region
• Responsibilities
• Task examples</td></tr>
<tr><td>Low-code development platforms</td><td>How did you begin using Microsoft Power Platform?
• What was the purpose for using it?
• Was the requirements clearly defined?
• Who asked you to do it?
• What was your experience?
• How did you learn more?

How did the course of the solution development go from a-z?
• How did you scope/define the solution?</td></tr>
</table>

	○ Who, what, when • How was the solution developed? ○ Who, what, when • How was the solution tested? ○ Who, what, when • How was the maintainance of the solution handled? ○ Who, what, when What other contexts have you used it in? • Client projects, internal projects, personal projects? • Were you approaching the development differently?
Governance of low-code development solutions	What thoughts did you make about governance of the solution? • Who should have access and who should not? • Who maintains the solution?
Final thoughts	Do you have any documentation on the solution/project that I can get access to? • SDD's • User guides • Project plans Do you have any final thoughts you would like to add?

Appendix 2 – Governance interview guide

I am investigating the use of low-code development platforms in businesses, with a special focus on the use of the Power Platform in EY. One of the areas I am investigating is the governance around such platforms, which is where you come in. Before we start the session I would like to ask you a couple of formality questions: - Can I record this session?

<table>
<tr><td colspan="2">

- Do you wish to stay anonymous?
- Would you like to have quotes send for approval?
- Do you have any questions or concerns you would like me to consider?

</td></tr>
<tr><td>Introduction</td><td>Can you tell me a little bit about yourself?
Name
Age
Title
Previous work
Education
What is your role at EY?
Title
Global/Area/region
Responsibilities
Task examples</td></tr>
<tr><td>Low-code development platforms</td><td>What is your general opinion on the low-code development platforms?
Benefits?
Challenges?
Opportunities?
How do you see the role of low-code development in the future?
Bigger?
Smaller?
Who develops using these tools?
How are low-code development platforms used in EY?
Dedicated teams?
Employee level use?
Innovation internally</td></tr>
</table>

Governance of low-code development platforms	How are you governing these platforms? • Who decides the rules? • How are the rules decided? • Technical capabilities? • What Business interests are considered? What considerations are you having regarding restriction or empowerment of the employee? • Security? • Information compartmentalization? • Training or certifications? • Business value? • Stakeholder management? How are you monitoring how these platforms are being used? • Admin dashboards? • Usage Reports? • What data do you have on a low-code solution? • Do you analyze the specific solutions?
Processes pertained to low-code development platforms	How is the process of taking a new low-code development platform into the business? • Approvals • Stakeholders • Evaluation How is the process for an employee to get started with using these Platforms? • Training? • System set-up • Development • Deployment

	How is the process for an employee to get access to more functionality in these platforms? • Ticket in a portal • Mail to a team • Evaluation and access How is the process for handling non-compliant uses of these tools? • Mail the owner of the solution • Evaluate the risk • Notify management
Final thoughts	Can you help me get in touch with… • Some user of low-code tools with a non-technical background? Do you have any final thoughts you would like to add?

*Grey section was not included for governance stakeholders

Appendix 3 – Codebook

Attached is the code book created from the inductive coding.

Appendix 4 – Interview transcript Sheldon

Appendix is attached as a separate file

Appendix 5 – Interview transcript Michael

Appendix is attached as separate file

Appendix 6 – Interview transcript BMC

Appendix is attached as separate file

Appendix 7 – Interview transcript InfoSec (Anonymous)

Appendix is attached as separate file

Appendix 8 – Interview transcript Vishal

Appendix is attached as separate file

Appendix 9 – Interview transcript Parulbahen

Appendix is attached as separate file

Appendix 10 – Interview transcript Aurimas

Appendix is attached as separate file

Appendix 11 – Interview transcript Vicki

Appendix is attached as separate file

Appendix 12 – Interview transcript Jose

Appendix is attached as separate file

Appendix 13 – Interview transcript Tim

Appendix is attached as separate file

Appendix 14 – Interview transcript Eoin part 1

Appendix is attached as separate file

Appendix 15 – Interview transcript Eoin part 2

Appendix is attached as separate file

Appendix 16 – Interview transcript Anbu

Appendix is attached as separate file

Appendix 17 – Interview transcript Security consultant (Anonymized)

Appendix is attached as separate file