The Best of Both Worlds: Combining Low-Code and Pro-Code for Governmental Process Optimization

W.L. van Brussel | 2082279

Tilburg School of Economics and Management, Tilburg University

390306-M-18: Master Thesis Information Management

Assistant Professor P.K. Medappa

June 6, 2025

Abstract

This thesis focuses on the strategic challenge that government organizations face when choosing between low-code and pro-code development methods for the implementation of applications. Although low-code technologies enable rapid development and cost savings, they often fall short in terms of robustness, scalability and integration, especially in complex IT environments. There is currently no structured framework that helps organizations make an effective choice between these technologies.

Based on the Design Science Research approach, this study has developed an assessment framework that categorizes applications based on two dimensions: development efficiency and execution efficiency. The model, inspired by the IT Strategic Impact Grid, was built on the basis of literature research and ten semi-structured interviews with IT professionals within the Netherlands Enterprise Agency.

The model divides applications into four quadrants, each linked to a suitable development strategy: low-code, pro-code or a hybrid form. The main conclusion of this research is that no single development method is universally superior, as the optimal approach depends on the specific characteristics of the application and the context.

This study contributes to the academic knowledge about hybrid software development and offers a practically applicable instrument for technological decision-making within the government. It underlines the importance of aligning the development strategy with the context, so that government organizations can digitize quickly and reliably.

Preface

In front of you lies my master thesis, written to complete the MSc Information Management program at Tilburg University. This research was conducted as part of my graduation trajectory at the program and focuses on the strategic use of low-code and pro-code technologies within the government.

I would like to thank my supervisor, assistant professor P.K. Medappa, for his expert guidance, critical questions and support during the trajectory. His insights have helped me to take the research to a higher level.

In addition, I am very grateful to the professionals of the Netherlands Enterprise Agency and other government services for their willingness to share their knowledge and experiences during the interviews. Their openness and expertise have added great value to this research.

Finally, I would like to thank my family for their support, patience and encouragement in the past months.

Contents

1. Introduction

1.1 Problem Relevance

In the current digital transformation, organizations are increasingly opting for low-code development platforms, which enables them to develop applications faster and at a lower cost (Cabot & Clariso, 2022). However, this rapid adoption is often accompanied by an overestimation of the possibilities of low-code, with traditional pro-code methods systematically being skipped in non-software companies. This can be problematic, as the robustness, flexibility and complexity required in professional IT environments are not sufficiently guaranteed. Moreover, recent research shows that the role of low-code within these professional contexts is still underexposed (Khalajzadeh & Grundy, 2024; Mosquera et al., 2024).

The lack of a clear and structured framework for making a well-considered choice between low-code and pro-code poses a crucial challenge. Organizations need guidelines that not only address development speed and cost savings, but also take into account the requirements for stability, security and IT governance. Low-code can enable faster and more flexible process integration, but at the same time requires careful coordination with existing, traditional systems (Buist, 2024), while clear IT Governance guidelines are necessary to ensure the control and security of the systems (Ajimati et al., 2024).

A hybrid approach, combining the benefits of low-code with the flexibility and control of pro-code, seems a promising solution. However, the successful implementation of such an approach requires clear frameworks, effective collaboration between modelers and programmers, and a deep understanding of which applications benefit optimally from which method.

This issue is particularly relevant in the context of government, where there is strong pressure to reduce both development costs and lead times, while simultaneously meeting high demands for stability, scalability, and legal compliance (OECD, 2021). The absence of clear

guidelines on when and how to use low-code or pro-code technologies hinders governmental organizations in making strategically responsible decisions that optimize both efficiency and effectiveness, which underlines the need for further research.

1.2 Research Objective

The aim of this research is to develop a framework that provides guidelines for the adoption of low-code and pro-code. The structure of this framework is based on the IT Strategic Impact Grid developed by Nolan and McFarlan (2005) and will help determine which applications in a government context are suitable for low-code and which can be better developed with pro-code. In addition, it will investigate how these technologies can optimally work together to maximize both efficiency and effectiveness. The research uses qualitative research in the form of semi-structured interviews with IT professionals to determine the axes of the grid, categorize the applications and validate the framework. The results of this research will contribute to both scientific insights and practical guidelines for organizations that want to use a hybrid development approach.

1.3 Research Question

The central research question is: *Which combined design of low-code and pro-code would optimize efficiency and effectiveness for implementing applications in a governmental environment?*

In order to systematically explore this research question, it is broken down into two guiding subquestions:

1. How can governmental applications be categorized?
2. For each application category, what specific combination of low-code and pro-code design would enhance both efficiency and effectiveness in implementation?

1.4 Research Approach

This research uses a Design Science Research approach, in which both the design of an artefact and the design process are central. In line with the guidelines of Walls et al. (1992), design science focuses on developing solution-oriented artefacts through an iterative process of design, evaluation and refinement. In this research, this methodology is applied to develop a framework that supports governmental organisations in making decisions about the use of low-code or pro-code in the development of applications.

The research consists of two main components: a literature study and an empirical study. The meta-requirements are to achieve higher efficiency and effectiveness in the development of applications. Based on these meta-requirements, relevant kernel theories are then analysed. These theories provide scientific support for the design of the framework and help define the design principles that contribute to realising the stated goals (Walls et al., 1992).

Based on the insights from the literature study, a first version of the framework is developed in the empirical part. This is done by means of seven semi-structured interviews with ICT professionals from the Dutch government, in which six different types of applications are discussed. These interviews include enterprise architects, analysts, developers, and program managers with direct experience in both low-code and pro-code environments. Their insights contribute to the conceptual elaboration of the framework's axes. The development process follows an iterative approach, in which new insights from the interviews are directly incorporated into the design decisions. The result of this phase is a framework that meets the requirements as a meta-design.

In order to guarantee the validity of the framework, it is evaluated by means of three validation interviews with a new group of experts, who were not involved in the initial development phase, thereby ensuring an independent assessment. This validation study focuses on testing the effectiveness and applicability of the framework in practice. This involves evaluating the extent to which the framework meets the established meta-requirements and

whether the proposed design principles actually contribute to the intended decision-making support (Hevner et al., 2004).

Due to this methodological design, the research is in line with the guidelines for Design Science Research. By combining ten detailed expert interviews, two phases of data collection, development and validation, and integration with kernel theory, the research ensures that the framework is both theoretically grounded and empirically validated. As such, the study adheres to the standards of Design Science Research and contributes a practically applicable artefact to the field (Walls et al., 1992).

1.5 Research Relevance

This research contributes to both scientific knowledge and practice. Both components are elaborated below, starting with academic relevance, followed by managerial relevance.

1.5.1 Academic Relevance

Within the existing academic literature, much attention has been paid to low-code as a means to accelerate software development, especially in the context of citizen development and simple business applications (Carroll & Maher, 2023; Overeem & Jansen, 2021). However, the application of low-code within professional IT environments, in which aspects such as complexity, scalability and reliability are crucial, has only been researched to a limited extent (Indamutsa et al., 2021). In particular, a theoretical framework is lacking that provides guidance on when low-code, pro-code or a combination of both is most suitable within a complex organizational context. This research attempts to fill this gap in the literature by analyzing how low-code and pro-code can complement each other and how this can be translated into a strategic decision model, inspired by the IT Strategic Impact Grid (Nolan & McFarlan, 2005).

In addition, the public sector is an underexposed area of application in the academic discussion on software development. Most existing models have been developed with

commercial applications in mind, in which flexibility and time-to-market are paramount (Baumgarten et al., 2024). These principles do not sufficiently meet the specific requirements of government organizations, such as stability, legal accountability and governance (OECD, 2021). By focusing on a public organizational context, this study contributes to expanding the knowledge about digitalization within governments, a domain that has received relatively little theoretical attention to date.

Finally, the study focuses on the concept of hybrid software development, which is gaining importance in practice, but has not yet been sufficiently evaluated in academic terms. By developing a classification model, this study aims to contribute to the further conceptualization and theory formation around hybrid development strategies.

1.5.2 Managerial Relevance

In practice, the need for guidance in technological choices within software development is also growing. Government organizations are in a transition phase in which traditional customized solutions are under pressure due to rising development costs, increasing lead times and a shortage of specialized programmers (Ewers & Vessey, 1981). At the same time, low-code platforms offer opportunities for acceleration and agility, but there is often a lack of clear guidelines on when and how this technology can be used effectively, while maintaining quality, control and scalability (Buist, 2024).

Specifically within government services such as the Netherlands Enterprise Agency (RVO), where a wide variety of digital processes exists, there is a need for guidance to systematically assess applications and strategically substantiate technological choices. This research responds to this need by developing a practically applicable model that structures and makes this consideration explicit. In this way, it supports decision-making about software architecture within complex government organizations.

1.6 Thesis Structure

This thesis is structured according to the methodological guidelines of Design Science Research, as formulated by Hevner et al. (2004). The research design is carefully designed to ensure both scientific and practical relevance and is in line with the need for technology-driven solutions within organizations.

Chapter 2, Literature Review, examines relevant theories, models and previous studies on the topic, providing the theoretical foundation and revealing knowledge gaps in the existing body of knowledge. Subsequently, the Research Methodology describes the research design, methodology and data collection methods used.

Design & Development outlines the development of the proposed artifact, including design decisions, implementation process and key features. The subsequent chapter, Demonstration & Evaluation, shows how the artefact is applied in the specific context of developing applications at RVO and then evaluates the effectiveness and performance of the artefact.

The Discussion reflects on the research results. Finally, the Conclusion summarizes the main findings, highlights the contributions to theory and practice, discusses limitations of this research and provides recommendations for further research.

2. Literature Review

This chapter discusses relevant literature on low-code and pro-code development, with a focus on the impact on efficiency and effectiveness within IT strategies. In addition, the model that serves as the foundation for the artifact is also discussed. In doing so, this chapter forms the scientific basis of the research and positions the study within the broader field of Information Systems Research.

2.1 Low-Code and Pro-Code

In recent years, new challenges have emerged that low-code technology can address as part of the digital transformation. These challenges include the increasing shortage of professional programmers (Carroll et al., 2021), the growing demand for shorter development times (Elshan et al., 2023), and the need for cost reduction (Fernandes et al., 2020). One of the biggest challenges within processes is the difficulty of communicating requirements effectively, because users cannot always formulate their precise needs well or because these change during the process (McLean, 1979).

Since the early 1980s, a form of citizen development has been described in the literature, originally intended to reduce the number of links in the development process. By enabling end users to develop their own applications, the so-called "maintenance problem" can be limited, namely adapting software to changing user and environmental requirements (Lientz et al., 1978). This can reduce time-consuming consultations and miscommunication (McLean, 1979).

Although citizen development was initially intended to directly involve end users in software development, this approach has evolved in recent years into the broader concept of low-code development. As a result, there is now no clear definition, despite the growing body of literature. Cabot and Clariso (2022) and Carroll and Maher (2023) describe low-code as software solutions that allow both non-technical employees and professional programmers,

often with the help of AI, to develop software. Previously, Cabot (2020) placed low-code within the broader context of model-based development, which is also referred to in other studies as Model-Driven Development (Overeem & Jansen, 2021). An example of this is the BPMN-based platform of Funk (2022), in which citizen developers can develop applications via diagrams and optionally supplement them with Python code for specific requirements. Wang et al. (2021) limit the definition of low-code to graphical models for citizen developers, which is classified by others as no-code development (Pantelimon et al., 2019). Since the terminological boundary between no-code and low-code is blurred in the literature, both concepts are included in this thesis under the heading of low-code development. In this study, low-code development is therefore defined as the model-driven development of software solutions, requiring minimal manual coding. This definition is therefore in line with the approach of Cabot (2020) and Overeem and Jansen (2021).

An increasing number of researchers and professionals, including Gartner, recognize the value of low-code technologies for digital transformation by citizen developers (Carroll et al., 2021; Hurlburt, 2021; Ajimati et al., 2024). Previous studies suggest that low-code technologies offer more benefits for citizen developers than for professional developers (Carroll & Maher, 2023; Indamutsa et al., 2021). However, Alsaadi et al. (2021) demonstrate that low-code can also be beneficial for professional developers in certain situations. The extent to which this is the case depends on the specific application and the complexity of the required functionalities.

2.2 Strategic Impact Grid as Structural Foundation

The Strategic Impact Grid by Nolan and McFarlan (2005) positions IT within organizations on the basis of two axes: the need for new information technology and the need for reliable information technology. Although these substantive dimensions are less applicable to this study, the formal structure of the model offers a clear and valid framework for

considerations, namely two axes that lead to four quadrants with corresponding recommendations.

This thesis adopts this structure, but revises it in terms of content with axes that better match the central research question. The adjusted axes are the need for development efficiency and the need for execution efficiency, which will be further elaborated below. Within the model, efficiency is expressed in the axes themselves, while effectiveness is pursued by applying the correct recommendation per quadrant: a low-code, pro-code or hybrid approach, depending on the context. This makes the model suitable as a practical framework for decision-making within government organizations.

2.3 Efficiency

Efficiency in software development concerns the speed and cost at which software solutions can be developed and implemented. Low-code offers advantages such as shorter development times and lower costs (Dong et al., 2024), whereas pro-code is more efficient for complex applications requiring customization. This leads to the observation that efficiency can be measured in various ways: time-based, effort-based, collaboration-based, and financially based (Ajimati et al., 2024). In order to properly plot efficiency in the model, it was decided to divide it into two main areas: the development process and software execution. In execution, time-based efficiency plays a role, whereas the other three forms relate to the development process.

Effort-based efficiency relates to the amount of labor required from a developer, particularly relevant for tasks such as programming or modeling applications (Pacheco et al., 2021; Pantelimon et al., 2019). Dunsmore and Gannon (1979) emphasize that programmer productivity is a crucial component of the evaluation process, as low productivity can result in significant hidden costs. Chrysler (1978) introduced a framework to examine six variables influencing programmer productivity, with Benbasat and Vessey (1980) later adding a seventh

variable. Various studies have sought ways to reduce effort, for instance, by structuring teams as agile as possible (Dikert et al., 2016) or by employing 'modern' software practices such as modularity (Lourenco et al., 2021), top-down development, and structured design (Zmud, 1980). Meador and Mezger (1984) argue that the most productive programming languages require the least effort, as they enable shorter and simpler code to achieve a goal. Harel and McLean (1985) point out that non-procedural languages, especially for novice programmers, accelerate application development. Furthermore, using logical, natural names instead of numerical codes enhances language accessibility and improves usability (Harris, 1984).

In recent years, attention has also been given to code generation via natural language queries in combination with artificial intelligence (Dong et al., 2024). Low-code platforms contribute to these productivity improvements by concealing as much code as possible (Metrolho et al., 2019) and instead utilizing a graphical interface (Wang & Wang, 2021). Research by Bexiga et al. (2020) demonstrates that the effort required for development using low-code platforms decreased from five to three days, enabling smaller companies to develop custom applications (Reilly, 2021) that were previously unaffordable (Moskal, 2021). By reducing the need for extensive coding, maintenance becomes simpler, and modifications can be implemented quickly without requiring complex code. The effort saved through low-code technology can be redirected by developers toward more complex tasks (Alsaadi et al., 2021).

Additionally, collaboration plays a significant role in the efficiency of software development. Collaboration efficiency refers to how well developers can collectively initiate and successfully complete software development tasks. (Pacheco et al., 2021). Zmud (1980) asserts that the primary causes of high costs and delays in software projects are not necessarily technical limitations but often result from poor management. According to Zmud (1980), the most significant issue is the incomplete or incorrect understanding of requirements, which makes it impossible to validate software correctly or manage the development process effectively. The use of graphical techniques can enhance project understanding, as they are

both concise and comprehensible. Low-code technologies contribute to collaboration efficiency, but they can also cause a lock-in effect, as platforms from different vendors vary significantly, making it challenging to transition between them (Schenkenfelder et al., 2023). This issue is compounded by the fragmented nature of platforms, which complicates uniform integration (Alsaadi et al., 2021; Hoogsteen & Borgman, 2022). Vendor support is often necessary for low-code technology, which can further complicate the development process. Meador and Mezger (1984) identified two key areas for evaluating vendor performance: the number of obstacles encountered during installation and upgrades, and the quality of support provided during product usage.

The third and final type of efficiency in the development process, financial-based efficiency, focuses on reducing costs associated with software development (Bhattacharyya & Kumar, 2021; Pichidtienthum et al., 2021). Since the 1980s, software costs have been a dominant factor in information systems, with predictions at the time that software and hardware expenditures would constitute 90% of costs by 1985 (Ewers & Vessey, 1981). Hitt and Brynjolfsson (1996) noted that IT provides clear benefits to consumers but has not yet had a significant impact on corporate profitability. Traditional development methods can incur high costs, particularly due to the scarcity of developers required to write complex code (Marek et al., 2021; Moskal, 2021). By leveraging low-code technologies, development time can be significantly reduced, thereby lowering software development costs (Asawa et al., 2021; Hoogsteen & Borgman, 2022).

Finally, time-based efficiency primarily concerns application performance during execution and is often referred to as operational efficiency (Pan et al., 2024) or computational efficiency (Op 't Land et al., 2021). Although there are various approaches to measuring this efficiency, no perfect method currently exists due to the limitations of available techniques (Pan et al., 2024).

During software execution, some performance optimizations remain possible through compilers, but manual optimization is still required to achieve substantial efficiency improvements (Pan et al., 2024). Previous studies, such as those by Harel and McLean (1985), have demonstrated that programming languages play a crucial role in the computational efficiency of programs. They found that procedural languages often outperform non-procedural languages in terms of execution time. Additionally, the distribution of workload between humans and machines is becoming increasingly important: while hardware is becoming more affordable, the cost of human programmers is rising. However, Harel and McLean (1985) caution that the benefits of computational efficiency can be lost if software requires frequent modifications. In such cases, the advantages of effort-based efficiency often outweigh the benefits of improved time-based efficiency (Harel & McLean, 1985).

With the rise of AI-assisted programming techniques, such as the use of large language models, automatic code generation is becoming more common. However, generated code is often less efficient than manually written code (Pan et al., 2024). At the same time, the efficiency of human-written code declines as problem complexity increases. This pattern is also observed in AI-generated code (Pan et al., 2024). A better understanding of coding patterns could enhance both manual and AI-based programming by improving the efficiency of both approaches (Cabot & Clariso, 2022).

Moreover, the transparency of generated code in low-code platforms is often seen as an issue. Developers have limited control over the actual code executed after modeling, making it difficult to optimize software behavior. The rise of AI in software development adds further complexity to this process. AI components interact with traditional software, yet these interactions are often difficult to trace and explain. Low-code platforms have so far paid little attention to this challenge, complicating AI integration within these environments. Additionally, AI models often utilize their own monitoring and analysis tools, further complicating management within a low-code environment (Cabot & Clariso, 2022).

2.4 Effectiveness

Meador and Mezger (1984) argue that two aspects are central when choosing software solutions: the productivity of the software in relation to the set goals and its accessibility for the intended users. This also holds true when deciding between low-code and pro-code approaches. A well-considered use of these development forms, tailored to context and purpose, largely determines the success of digital solutions.

As such, low-code platforms are widely recognized for their ability to support rapid application development and provide accessibility to non-technical users (Alsaadi et al., 2021; Magesa & Jonathan, 2021). This makes them particularly effective in situations where speed, user engagement, and collaboration between business and IT are key. The visual development environments and standard components allow for shorter development cycles and easier customization, increasing agility and making functionality more responsive to user needs (Reilly, 2021; Zmud, 1980). In addition, IT costs can be reduced (Schenkenfelder et al., 2023) and visual modeling helps reduce miscommunication between business and IT (Elshan et al., 2023; Zmud, 1980).

It is crucial that systems are interactive and user-friendly when end users also become developers (McLean, 1979). Tanimoto (2013) emphasizes that a dynamic development environment contributes to progress. For example, DMN decision tables, which are similar to spreadsheets in terms of layout, are more likely to be accepted than BPMN diagrams (Funk, 2022). However, Funk (2022) prefers the principle of "As Much Code As Needed", where users do not have to learn more than one programming language.

However, it remains uncertain whether these advantages actually lead to structural business improvements (Cabot, 2020). A major disadvantage of low-code development platforms is the strong lock-in of third parties. Due to high investment costs and limited portability of developed applications to other platforms, there is a dependency on suppliers (Ajimati et al., 2024). Furthermore, the abstraction of programming code within these platforms

creates potential security risks. Inexperienced citizen developers can unintentionally develop vulnerable applications without full insight into the underlying risks (Oltrogge et al., 2018).

Pro-code development also proves to be more effective in situations where high performance, reliability, or security are required. The control that pro-code provides over architecture, logic, and system interaction enables developers to optimize software down to the smallest detail (Zmud, 1980). This is especially important in large-scale or critical systems, where even small performance losses can have major operational consequences (Mosquera et al., 2024). Although development paths are typically longer and entail higher costs, these are more than offset by benefits in scalability, maintainability, and auditability (Ajimati et al., 2024; Marek et al., 2021). Full ownership of the code base also facilitates compliance and long-term maintenance (Dong et al., 2024).

Increasingly, the effectiveness of software development is sought in hybrid models that combine low-code and pro-code. Within such strategies, low-code is used for generic components such as user interfaces or workflow support, while pro-code is applied to components that require customization or high computing power (Buist, 2024; Metrolho et al., 2019). Modern low-code platforms offer integration options that support collaboration with pro-code, increasing effectiveness in environments where flexibility and performance must be combined (Dong et al., 2024). However, this effectiveness depends on careful management of interfaces and modular separation of responsibilities. If this is not done sufficiently, technical issues can arise that harm maintainability and reliability (Hoogsteen & Borgman, 2022; Pan et al., 2024). The use of AI-supported development tools within both low-code and pro-code can further complicate this situation. Although such tools increase development speed, the processing is often less transparent, which can damage confidence in the generated solutions (Cabot & Clariso, 2022) and lead to significantly lower execution efficiency (Pan et al., 2024).

2.5 Conclusion

This chapter has provided insight into the possibilities and limitations of low-code and pro-code development, with special attention to their impact on efficiency and effectiveness within IT strategies.

Low-code technologies address key challenges in software development, such as time pressure, high costs, and the shortage of programmers. By reducing manual coding and using visual models, they mainly improve efficiency in relatively simple applications. Pro-code, on the other hand, remains indispensable for complex, secure and scalable systems, in which customization and control are essential. The effectiveness of both approaches depends strongly on the context and purpose of the software. Hybrid models are increasingly being used, in which low-code and pro-code are combined to utilize the advantages of both. However, this requires careful coordination to prevent technical issues.

The Strategic Impact Grid (Nolan & McFarlan, 2005) was used in this research as a structural basis for the assessment framework to be developed. The axes of the model have been adjusted in terms of content to better match the research question, which increases the applicability within a government context. Based on this literature, this chapter provides a strong foundation for the design of the artifact in the next phase of the research.

3. Research Methodology

3.1 Research Context

The Netherlands Enterprise Agency (RVO) is an executive agency of the Dutch government that supports entrepreneurs in realizing sustainable and innovative ambitions. An important part of the service provision is the development and implementation of subsidy applications, financing schemes and international trade promotion. Currently, these subsidy applications run on the Basic Administration System (BAS), which has reached the end of its life cycle (IV architecture team, 2023). As a result, a migration to new implementation platforms is inevitable, while there are no clear guidelines yet for the distribution of governmental processes across various technological solutions.

RVO is faced with the choice between using low-code technologies, such as Blueriq within the Implementation Platform Netherlands (UPNL), or traditional pro-code solutions such as Java (Van Deenen, 2024; IV architecture team, 2023). This choice is crucial, as RVO's applications must not only function efficiently and flexibly, but must also guarantee stability, security and reliability (Legtenberg, 2024). The lack of scientific basis makes it difficult to make well-considered decisions about the correct use of low-code and pro-code within RVO.

3.2 Data Collection

This research uses a qualitative and comparative research design, combining semi-structured interviews with IT professionals and a comparative case study on the implementation of subsidy applications. By combining practical experiences and expert insights, this research aims to develop a framework that supports government organizations in making substantiated choices around the use of low-code and pro-code in application development.

A total of ten semi-structured interviews were conducted with IT professionals within the Dutch government. The research started with an interview with an enterprise architect at RVO,

focusing on the strategic allocation of applications across different development platforms (interview 1). Subsequently, an analyst involved in the migration of an outdated low-code platform was interviewed, to gain insight into the criteria for the allocation of processes to new platforms (interview 2).

After this, multiple interviews were conducted with modelers of different subsidy applications: two modelers of comparable structural subsidy applications (interview 3), a modeler of two crisis subsidy applications (interview 4) and a modeler of a smaller subsidy application (interview 5). Subsequently, an interview was held with a programmer and team manager responsible for an application developed in pro-code, to find out why low-code was deviated from in this case (interview 6). The series was continued with a conversation with the lead analyst of the largest low-code platform within RVO, in which the variation in subsidy applications built with no-code were discussed (interview 7). These conversations provided detailed input for placing the applications within the developed model.

Next, a discussion was held with a solution architect and program manager who are involved in the exploration of a new development platform, to test the strategic applicability of the model (interview 8). Additionally, an external architect from another government organization was interviewed, where pro-code is the standard for software development but low-code is now also being adopted for specific processes (interview 9). Finally, the results of all interviews were evaluated and discussed with the architect of RVO's largest low-code platform, to validate both the model and the research results. (interview 10).

The interviews were conducted using a semi-structured approach, which on the one hand allowed for a certain degree of comparability in the answers, and on the other hand left room for in-depth and context-specific insights. All respondents were informed in advance about the purpose of the research and agreed to participate. Anonymity was offered where desired. The conversations were recorded and transcribed to ensure the reliability of the analysis. As

Dutch is the working language within the Dutch government, all interviews were conducted in Dutch. Transcripts can be made available upon request.

In order to strengthen the reliability of the model, the insights from the interviews were tested with another government service where pro-code software development is the norm (interview 9). This makes it possible to place the results of the RVO case in a broader context, which benefits the applicability of the final model. An overview of the interviews is included in Table 1.

Table 1

Overview of the conducted interviews

Interview number	Function	Objective
Interview 1	Enterprise architect (A)	Current allocation of applications across different development platforms
Interview 2	Analyst (B)	Migration to new low-code platform
Interview 3	2x Modelers (C, D)	Characteristics structural subsidy processes
Interview 4	Modeler (E)	Characteristics crisis subsidy processes
Interview 5	Modeler (F)	Characteristics small subsidy process
Interview 6	Programmer (G) & Team manager (H)	Characteristics tender program (pro-code)
Interview 7	Lead analyst (I)	Characteristics no-code subsidy processes
Interview 8	Solution architect (J) & Program manager (K)	Exploration of a new development platform, validate outcome
Interview 9	Architect (L)	External governmental organization, validate outcome
Interview 10	Architect (M)	Validate outcome

4. Design & Development

This chapter outlines the development of the proposed artefact, including the design decisions, the implementation process and the main features. First, in subchapter 4.1 the features that lead to a classification in the model will be discussed. Then, in 4.2 the resulting quadrants will be discussed and linked to the literature to recommend a research method for each of the quadrants.

4.1 Application Classification Characteristics

To determine whether low-code, pro-code or a hybrid combination is the most suitable development approach for a specific application, a two-dimensional assessment model has been created consisting of two axes: need for development efficiency and need for execution efficiency, based on the four ways in which efficiency can be measured, as observed by Ajimati et al. (2024). Each axis consists of four quantitative characteristics, which together provide a weighted picture of the development and execution characteristics of the application. All characteristics are individually scored on a scale of 0 to 10. The total score per axis is then calculated as the arithmetic mean of the four underlying characteristics.

4.1.1 Need for Development Efficiency

The need for development efficiency axis includes characteristics related to the development speed, technical complexity and manageability of the application. The following four characteristics form the basis of this axis:

First, the time-to-market characteristic describes the speed at which the application must be realized (Mosquera et al., 2024). A high score indicates a strong need for fast delivery, such as due to policy pressure or urgency, while a low score indicates a less critical timeline.

In addition, the extent to which the application depends on integrations with external systems is assessed. External integrations entail additional development work, dependencies

and error sensitivity, and can increase the complexity of deployment and maintenance (Alsaadi et al., 2021; Hoogsteen & Borgman, 2022). A high score on this characteristic means that the application functions largely autonomously with no or only limited external connection. A low score indicates an application that requires multiple integrations with external services, systems or data endpoints.

The complexity and customization characteristic describes the extent to which the functional logic of the application deviates and requires customization beyond standard solutions (Metrolho et al., 2019; Moskal, 2021). This concerns the internal technical and functional complexity, such as specific business rules, exception structures, conditional logic or data manipulations that do not fit directly within standard components. A high score on this characteristic indicates a relatively simple and generic application that requires little to no customization. Applications with a high degree of complexity, domain-specific logic or unique requirements, on the other hand, score lower, because they benefit less from standard solutions, which limit the ability to develop efficiently (Ajimati et al., 2024).

Although both customization and external links contribute to the technical complexity of an application, these characteristics focus on different aspects. The complexity and customization characteristic describes the internal logic and functionality that must be developed specifically, while the dependency on external systems relates to the external integration burden (Alsaadi et al., 2021; Cabot & Clariso, 2022). These aspects can vary independently of each other. For this reason, both characteristics are included separately in the scoring system, so that nuances in technical burden and development strategy can be accurately analyzed.

The last characteristic influencing the need for development efficiency is the expected lifespan. This characteristic indicates how long the application is expected to remain in use. Applications with a short lifespan, such as temporary dashboards or one-off forms, can often be realized more quickly. A low score means a long-term commitment, in which aspects such as

maintainability and scalability weigh more heavily. A longer lifespan usually requires a more robust and future-proof architecture reducing the potential to optimize for development efficiency (Di Ruscio et al., 2022). Applications with a longer lifespan can also suffer more from lock-in if a switch to another technology or supplier is desired after a period, because there is limited transferability of processes (Ajimati et al., 2024).

4.1.2 Need for Execution Efficiency

The need for execution efficiency axis reflects how critical it is that the application operates with high performance, scalability and reliability under operational conditions. The four characteristics that form the basis of this axis are as follows:

The first characteristic, automatic handling, indicates the degree of autonomy of the system. Applications that mainly serve to support manual processes score lower, while systems that use fully automated workflows score higher due to their higher requirements for stability, error handling, reliability and performance (Pan et al., 2024).

In addition, the expected number of end users gives rise to increased requirements for execution speed and scalability. A large group of simultaneous or frequent users requires a system that performs continuously and reliably, even under peak load (Meador & Mezger, 1984). In such cases, efficient execution is essential. A low score means that the application is only used by a limited group of users, which means that there is less pressure on optimal performance or scalability.

Concurrent processing measures the expected level of parallel use or processing. With high concurrency, systems must be able to process multiple requests simultaneously without loss of performance, which places heavier demands on architecture and implementation (Pan et al., 2024).

The final feature is whether the application contains components with a high computational load, such as algorithms, calculations, machine learning or large data processing.

Such components require fast, stable and efficient execution, which results in a high score (Dong et al., 2024; Pan et al., 2024). If the application mainly consists of simple processing without heavy computational logic, then the execution efficiency is less critical and a lower score is awarded.

4.1.3 Delineation of context-specific characteristics

Within the scoring system used, these eight distinguishing characteristics were chosen, divided over the above-mentioned two axes. In broader literature, other factors are also mentioned that can influence the choice between low-code and pro-code, such as compliance requirements, security levels, and standardization of infrastructure (Ajimati et al., 2024). In the government organization studied, however, these characteristics are largely uniform: all applications fall under the same security and compliance policy, and are hosted within a centrally managed infrastructure.

For this reason, such characteristics were explicitly disregarded in the differentiation between applications. This increases the internal validity of the scoring system, without ignoring the context-specific boundary conditions. The focus is therefore on characteristics that do vary within the cases studied, and that are decisive in practice in the choice between a low-code or pro-code approach.

4.1.3 Overview of the Classification Characteristics

The previously discussed characteristics are summarized below in an overview table. Table 2 shows per characteristic how it contributes to the need for development efficiency or execution efficiency, as used in the classification model. This makes it possible to position applications systematically within the two-dimensional model.

Table 2

Overview classification characteristics

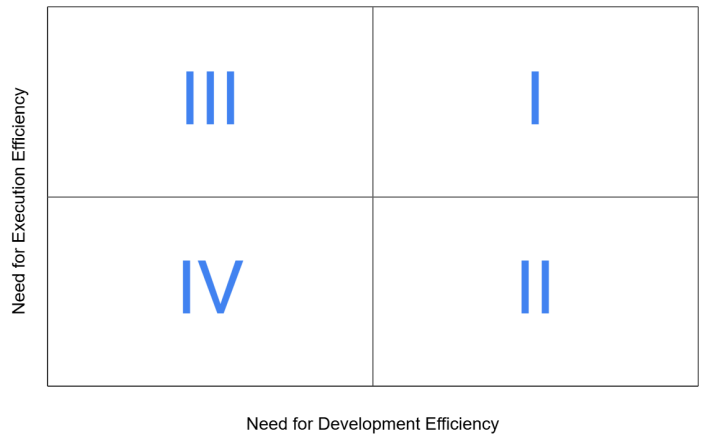
Axis	Characteristic	Description	Interpretation of score
Need for Development Efficiency	Time-to-market	Urgency and speed with which the application must be delivered.	A high score reflects high urgency and thus a stronger need for development efficiency.
	Integrations with external systems	Degree of dependence on external systems or connections.	A high score reflects low integration complexity, which enables more efficient development.
	Customizations	How much functional or technical customization is required.	A high score indicates low customization, allowing for more development efficiency.
	Expected lifespan	Expected duration of use of the application.	A high score reflects a short lifespan, where rapid and efficient development is more desirable.
Need for Execution Efficiency	Automatic handling	Degree of required automatic handling of processes within the application.	A high score reflects high automation, increasing demands on execution efficiency.
	Expected users	Expected scale of usage in terms of number of users.	A high score indicates high expected usage, requiring stable and scalable execution.
	Concurrent processing	Expected degree of parallel usage or concurrent processes.	A high score indicates high concurrency, necessitating a robust and efficient execution environment.
	Computational load	Degree of heavy calculations or data processing within the application.	A high score reflects high computational demand, which requires greater execution efficiency.

4.2 Quadrants

By entering the application scores on both axes, development efficiency and execution efficiency, into the two-dimensional model, a division into four quadrants is created (see Figure 1). Each of these quadrants represents a specific combination of development and execution properties, and thus provides direction for the choice of a suitable development approach.

Figure 1

Identified Quadrants



4.2.1 Quadrant I: High Development & High Execution Efficiency Required

In the first quadrant, both rapid development and excellent performance are crucial. The emphasis on development efficiency points to a context in which rapid delivery is essential. At the same time, stringent demands are placed on the execution quality of the software, such as real-time data processing or supporting large-scale applications.

Although low-code technologies offer significant advantages in terms of development speed (Alsaadi et al., 2021), they have shown limitations when detailed control over execution and in-depth optimization are required (Pan et al., 2024). In such contexts, a traditional pro-code approach is generally preferable. By using techniques such as modular architecture, optimized

programming practices, and direct access to underlying system resources, customized solutions can be realized that meet both speed and performance requirements (Zmud, 1980).

In many of these cases, a hybrid development strategy, combining low-code and pro-code, can also offer significant advantages (Dong et al., 2024). Within such an approach, low-code is primarily used for the development of standard functionalities, like workflow support, while pro-code is reserved for the critical performance-intensive components, such as complex calculation modules or backend optimizations (Buist, 2024). Modern low-code platforms facilitate this hybrid form by offering integration options, which enables development teams to maximize development speed without compromising the required implementation quality (Metrolho et al, 2019). However, the hybrid approach is not without risks: carefully delineating and managing the interfaces between low-code and pro-code components is essential to prevent the build-up of technical debt and to ensure the long-term maintainability of the software (Hoogsteen & Borgman, 2022).

In summary, it can be said that, if applied carefully, a hybrid development approach within this quadrant combines the best of both worlds: the speed and cost-efficiency of low-code development and the execution power and flexibility of pro-code.

4.2.2 Quadrant II: High Development & Low Execution Efficiency Required

Within quadrant II, rapid development is central, while the requirements for implementation quality are relatively low. This context is particularly relevant for applications such as internal tools, temporary solutions or process automations, where user-friendliness and speed of delivery prevail over performance or scalability.

Low-code platforms are an extremely suitable choice within this setting. The visual development environment, the possibility of rapid iterations and the accessibility for both professional developers and 'citizen developers' ensure that development cycles are

significantly shortened (McLean, 1979). These properties are particularly valuable in dynamic environments where flexibility and time-to-market are the highest priority.

The limitations of low-code with regard to implementation efficiency are not an obstacle here, since the applications generally do not operate under heavy loads and are mainly used internally. In addition, the standardized interfaces and visual modeling of low-code platforms help reduce miscommunications regarding functional requirements, which significantly improves collaboration between business and IT (Pacheco et al., 2021; Zmud, 1980). In summary, low-code development in quadrant II optimally meets the need for speed and agility, without the limited performance requirements being an obstacle.

4.2.3 Quadrant III: Low Development & High Execution Efficiency Required

Within quadrant III, the emphasis has shifted to optimal execution quality, while speed of development is subordinate. This type of context is common in critical applications and large-scale data processing platforms. In such cases, a traditional pro-code approach is essential. Low-code platforms fall short here due to their high level of abstraction, making the necessary fine-tuning optimizations impossible (Mosquera et al., 2024). In addition, a risk in terms of quality can arise if the speed of low-code techniques is prioritized too much (Ajimati et al., 2024). Although the lower development efficiency of pro-code development leads to longer development paths and higher initial costs, these can be more than compensated by the resulting benefits, such as faster response times, higher reliability and lower operational costs over the entire system lifecycle (Ajimati et al., 2024; Marek et al., 2021). In addition, full code ownership facilitates extensive audits, security optimization, and structured long-term maintenance (Dong et al., 2024).

In conclusion, in situations where performance and reliability are the highest priority, a pro-code development approach is indispensable, even at the cost of longer development time and higher complexity.

4.2.4 Quadrant IV: Low Development & Low Execution Efficiency Required

Quadrant IV is characterized by an environment in which neither rapid development nor high execution quality is required. Although development speed is not critical, the low complexity of low-code platforms reduces development effort and facilitates the delivery of functioning prototypes or simple applications without significant investment of time or resources (Metrolho et al., 2019; Reilly, 2021). This allows projects with limited resources to still achieve a workable result.

In contrast, a traditional pro-code approach is rarely justified here: the relatively high costs and longer development paths (Marek et al., 2021) are not in proportion to the limited functional and performance requirements. In addition, the use of low-code simplifies future iterations and maintenance, which is particularly beneficial in environments where flexibility is more important than technical perfection (Alsaadi et al., 2021).

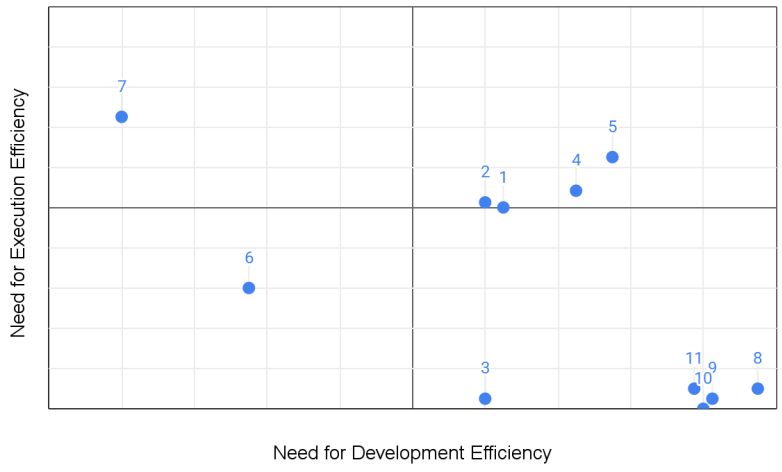
Overall, low-code in quadrant IV offers a cost-effective and sensible approach that meets the minimum requirements without introducing unnecessary complexity.

5. Demonstration & Evaluation

With the description of the design of the two-dimensional assessment model and the elaboration of the four quadrants, Chapter 4 has laid a solid theoretical and methodological foundation. In this chapter, the design is put into practice. Using interviews and concrete case studies, we test the applicability and reliability of the model within the context of governmental processes (see Figure 2). This shifts the focus from the abstract design to the empirical validation and evaluation of the developed artefact, based on the average of the underlying characteristics. In this way, it can be determined to what extent the proposed combination of low-code and pro-code actually contributes to efficiency and effectiveness in a government environment. Each quadrant is discussed separately, including the associated case, reflection and conclusion. Finally, the overarching findings were tested in an interview with two architects.

Figure 2

Distribution of the Cases



5.1 Quadrant I: High Development & High Execution Efficiency

As explained in Chapter 4, the first quadrant is characterized by the need for both rapid development and high execution efficiency. The analysis focuses on two types of cases, namely

crisis scheme applications and structural scheme applications. This section describes the characteristics of these cases and how they position themselves within this quadrant.

5.1.1 Practical Cases: Crisis Subsidy Schemes

Schemes 4 and 5 were introduced during the energy crisis and the corona crisis respectively to quickly support entrepreneurs financially. For both schemes, an application portal had to be available within a very short time, with an expected high application pressure of an estimated 60,000 users for scheme 4 and a total of around half a million applications over nine rounds for scheme 5. The need for speed led to the use of a low-code platform. With the consultation of multiple external sources and the necessary automatic processing of the applications of 66% and 50% respectively, the performance requirements were also high.

According to respondent E, modeller for both schemes, low-code made it possible to “create something that works in three seconds”, while traditional programming often takes many times as long. It is also indicated that many peripheral matters, such as error handling, data storage and screen structure, are ‘included’ in the platform. This allowed the development team, consisting of five modelers and two testers, to continue developing at a high pace, with testers even indicating that they had difficulty keeping up, according to respondent E.

The flexible design did not mean that the technology was without limitations. As soon as deviating or customer-friendly logic is required in the user interface, such as a complex interactive feature, low-code proves to be less agile and becomes dependent on supplier support or custom development, which leads to longer processes, indicated respondent E. For example, in the case of scheme 5, a dropdown component was confronted with scaling problems when the number of items to be loaded increased. In response, the supplier of the low-code technology had to develop a new component, which made it part of the low-code platform again, according to respondent E. At the same time, this shows that low-code platforms can grow and adapt to new requirements, provided that this fits within the architecture.

Another point of attention is the limited support for advanced integration requirements. Respondent E indicated that blueriq can call and process APIs, but offers little standard support for things like caching, transformations or ETL tasks, elements that are often essential for connections with external systems, as is necessary for scheme 4. As respondent E indicated: “You simply have ETL tools for that, but that is actually separate from the application.”

Although the application process for both schemes was linear and uniform, making it easy to model, it became clear that success partly depended on the collaboration between modelers and the business. According to several interviewees, the bottleneck was rarely in the technology, but rather in the timely and clear specification by the business. Where traditional software development often requires an iterative development process per functionality, working with low-code requires a different mindset in which standardization and reuse are central (Mosquera et al., 2024). This also requires a different division of roles and communication between domain experts and developers.

Furthermore, it was noted that low-code, due to its low threshold, entails the risk that inexperienced developers, such as juniors or mediors, are deployed on complex projects, under the motto that “everyone can do Blueriq”, respondent E indicated. This is a major pitfall: despite the simplified modeling process, architectural thinking remains necessary to keep the applications sustainable and maintainable.

Finally, it should be noted that these temporary arrangements provide insight into the power of low-code for urgent and short-term projects. As long as the process can be standardized well and the functional requirements are clear in broad terms, low-code appears to be a suitable technology to deliver a working application within a few weeks.

5.1.2 Practical Cases: Structural Subsidy Schemes

In contrast to the temporary nature of the crisis schemes is the structural and ongoing nature of the structural schemes that fall into this quadrant. These schemes, 1 and 2, have

existed for decades but have also been implemented via a low-code architecture for several years.

Scheme 1 supports approximately 35,000 applications annually and processes them partly automatically. Scheme 2 is comparable to scheme 1, but has a longer available time-to-market and supports approximately 25,000 applications. According to respondent C, 60–70% of applications are currently automatically decided positively on the basis of a prior classification. This classification, a decision model that determines whether an application meets the conditions, is not mathematically complex in terms of content, but does contain many logical checks and data links. Multiple data are retrieved and compared per application, including previously requested subsidy amounts, use of reporting codes that can change annually, and legal limits per company. Many of these controls require tables to be linked via keys, sometimes across different databases, which makes the system a relatively data-sensitive application.

The logic underlying this can be modelled well in a low-code environment, but does impose requirements on the performance of underlying databases and the degree of standardisation within data structures. Integrations with legacy systems are regularly performed. The technical complexity is therefore less in the computing power of the application, and more in the data volumes, the frequency of updates, and the interaction with external systems.

An important aspect that came up in the interviews is that this application is not ‘finished in one go’. Since going live, the schemes have been developed step by step. This concerns a development cycle of several years, in which choices about functionality and scope are made gradually, partly depending on available budgets. In contrast to the crisis schemes, where going live led to a conclusion of development, these schemes involve continuous expansion and refinement.

As with the other case, the success of low-code also appears to depend on the collaboration between content experts, the business, and modelers. The separation between

specifying and implementing blurs in this context, where the modeler not only ‘builds’, but also advises on screen layout, process logic, and user interaction. Reference is made to the communication skills of modelers, which, according to respondent E, often fit better with multidisciplinary collaboration than traditional developers.

5.1.3 Practice and Model

The interviews clearly show that the speed of low-code development is experienced as a decisive advantage. A working application could be delivered within a few weeks, which would not have been feasible in traditional programming environments. Respondent D, for example, stated: “You really don’t get any faster with programming, I am convinced of that.” At the same time, limitations of low-code platforms become apparent when there are performance-intensive components or exceptional functional requirements. A concrete example concerns the dropdown in arrangement 5 with a large number of items, which was initially not possible within Blueriq. Only after the development of a new specific component could this be adequately resolved.

This practice is in line with the literature, which states that low-code technologies are ideally suited for the development of standard functionalities, but fall short when advanced optimizations or in-depth system interventions are required (Cabot & Clariso, 2022). The need for real-time data processing, complex classification logic and heavy couplings, such as in structural arrangements 1 and 2, illustrates that low-code alone is insufficient to meet all performance requirements. A hybrid development strategy offers a solution here.

The integration of low-code and pro-code components is already being cautiously applied in practice. Although Blueriq itself is limited in data management and transformations, this is accommodated with external ETL tools and linked APIs. As respondent C emphasized: "You just have to have a decent application for that that manages that for you." This creates a clear division of tasks between the low-code platform for process logic and interfaces and

specialist tools or custom modules for performance-critical tasks. Literature confirms the effectiveness of this hybrid approach, provided that sufficient attention is paid to interface management and coordination between components (Hoogsteen & Borgman, 2022).

In addition, vendor lock-in is an important point of attention in the structural arrangements (Legtenberg, 2024). Because schemes 1 and 2 have been developed within the same low-code platform for years, dependency on both the technology and the supplier arises. By using a hybrid application, the performance-critical parts can be kept in-house by means of pro-code and this risk can be reduced.

Finally, the dangers of overestimating low-code simplicity by management were pointed out. “That is quite a pitfall of low-code, that managers think: we will put a medior and a junior in place and then everything will be fine,” according to respondent D. This underlines the importance of a professional architectural vision, even within a seemingly simple platform.

In conclusion, a pure low-code or pro-code approach seems insufficient within the quadrant where both speed and performance are important. A hybrid development strategy, in which standard components are quickly realized via low-code and performance-critical elements are optimized via pro-code, appears to be the most suitable solution in both theory and practice. This approach does require technical discipline and cooperation between development roles, in order to avoid technical debt and maintenance problems.

5.2 Quadrant II: High Development & Low Execution Efficiency

Quadrant II is characterized by an emphasis on rapid development, while the requirements for implementation quality remain relatively limited. In this context, low-code development fits well with applications as internal tools and temporary solutions. This section discusses the case of the adjustment concept within the RVO, and how it positions itself within this quadrant.

5.2.1 Practical Cases: Adjustment Schemes

Within the RVO, the adjustment concept is used for the management of various schemes, where each scheme consists of a set of parameters that are adjusted in a central case system. These schemes correspond most closely to what is also referred to in the literature as no-code, because no programming knowledge is required (Hurlburt, 2021). The platform does not support advanced automation, but does offer generic functionality that can easily be adapted to the needs of different schemes.

The emphasis is on "employee in-control", which means that employees remain responsible for the assessment, according to respondent I. There is only a limited degree of automation. Although some generic checks and links with external systems are possible, the assessment process is mainly carried out manually. The platform only supports automatic assignment and categorization functions, but these processes remain global and not specifically tailored to the content of the schemes. This makes the system flexible for a wide range of schemes, without requiring customization for each specific scheme. The variation in the number of cases and the complexity of the assessment is reflected in the number of employees involved in the process. In some cases, "you only have two people doing the whole scheme", while in other cases a more extensive evaluation takes place that requires multiple employees, respondent I indicates.

In terms of lead time, the technical setup of a new scheme can be completed quickly, but the actual lead time is often determined by the time the business needs to check the content of the scheme. The opening of schemes usually takes place in annual cycles and is repeated when a scheme is reopened, with only a few changes being made to the parameters.

Due to the low technical complexity and the limited number of applications and customization required per scheme, the adjustment concept is a typical example of a scenario in which low-code technology could function ideally. The implementation of the system is fast and accessible, and the platform is suitable for the limited requirements of the schemes themselves.

5.2.2 Practice and Model

The control concept fits well within this quadrant, as it is all about rapid development and limited complexity, with relatively low demands on the final execution quality. As expected for this quadrant, the emphasis is on speed and flexibility, and the emphasis on the quality of the execution infrastructure is minimal. The control schemes are generally not heavily loaded in terms of data traffic or user interaction, which makes the choice for a low-code solution logical.

Theory supports the use of low-code technologies in situations like this, where the speed of delivery and the simplicity of implementation outweigh the technical perfection of the final product (Bexiga et al., 2020; Wang & Wang, 2021). The low demands on performance and the possibility to quickly adapt the system to the specific requirements of a scheme make low-code ideal for this type of use (Metrolho et al., 2019). Therefore, no heavy technological infrastructure is required, which further accelerates the process.

In short, the case of the adjustment concept clearly shows how low-code technology can be optimally used in situations with low complexity and high demands on speed of delivery. The use of low-code enables organizations to respond quickly to changing requirements without this being at the expense of the necessary control and evaluation processes.

5.3 Quadrant III: Low Development & High Execution Efficiency

Quadrant III is characterized by a strong emphasis on quality of execution, while speed of development is of secondary importance. In this context, a pro-code approach is usually necessary due to the need for control, reliability and long-term manageability. This section discusses the case of a procurement application, and how it positions itself within this quadrant.

5.3.1 Practical Case: Tender Application

Application 7 illustrates a clear positioning within this quadrant. As a central government procurement application, built in Java, the platform requires full control over logic, data

structures and connections. The choice for a pro-code approach has partly grown historically, but is also mainly driven by the extreme variation and rule-driven complexity of the domain. With over 1200 possible procurement configurations and a constantly changing legal context, a high level of adaptability is essential. This flexibility has been achieved through a modular architecture, a metamodel layer and consistent code conventions within the team.

Although the technical calculations are not exceptionally complex, the number of database queries and dependency between input fields is high. This makes customization inevitable, which can be arranged well in a pro-code environment: "We can adjust every line of code that exists," according to respondent G. By developing all the logic ourselves, the team can optimize very specifically for performance, maintainability and security. The choice for Java is partly dictated by the maturity of the ecosystem and the level of knowledge present within the team. Respondent H also indicates that Java has the advantage that it is a direction with many available developers and that finding low-code developers is difficult due to the different platforms.

Interestingly, despite the low development speed that is usually attributed to pro-code (Alsaadi et al., 2021), the team is able to implement major changes quickly. This paradox is explained by the strong internal standardization and team continuity. However, it is important for this that the same 'dialect' is spoken in the code, says respondent G. By rebuilding the entire platform with uniform principles, extensions such as new European regulations can be implemented relatively quickly, provided that there is structural discipline.

5.3.2 Practice and Model

This tender program confirms the characteristics of the quadrant. The pro-code approach proves necessary due to the high degree of rule-specific logic, flexibility requirements and connections with external systems. The emphasis on execution quality is reflected in precise control over the data model, the application behavior and the long-term maintainability

(Meador & Mezger, 1984). Although the required development speed is initially less relevant, the case shows that it can increase if the team works according to uniform structures and principles.

5.4 Quadrant IV: Low Development & Low Execution Efficiency

Finally, quadrant IV includes situations in which both the development pressure and the demands on technical implementation are low. In such contexts, low-code is usually the most logical choice due to its low costs, limited complexity and high flexibility. This section discusses the case of a scheme within the RVO, in which the exceptional complexity and limited use lead to an unusual positioning within this quadrant.

5.4.1 Practical Case: Exceptional Subsidy Application

Subsidy application 6 is a special case within RVO: a subsidy programme of enormous financial size that is characterised by a low application frequency, but at the same time an exceptionally high complexity. Each application represents potentially millions of euros, and often has very specific, unique conditions and calculations. The scheme supports large-scale projects and requires complex calculation logic, exception handling and frequent policy adjustments. According to respondent J, the scheme is so extensive that it “actually consists of 18 schemes”.

The scheme is currently running on a legacy platform and must move to another platform within a few years. An attempt was made to map out the scheme for this purpose. An initial analysis of the scheme took almost half a year, during which a flood of exceptions, customised constructions and calculation rules were encountered. Respondent E stated: “Until the end of the analysis, we only received more questions... how does this work, and how does that work? There is simply a lot in it.”

Within RVO's multi-platform strategy, existing low-code platforms such as Blueriq or APEX have been considered as possible technologies. However, the functionality of the current low-code platforms falls short for the required complexity of this application. Adjustments would be so drastic that one expects to have to build an enormous amount of customization on a platform that is not intended for that purpose. At the same time, a full pro-code approach seems difficult to justify due to the limited frequency of use and the small number of users, especially in relation to the development costs. This puts this scheme in a technological no-man's land. As respondent K noted: "With the standard way in which we run most subsidy processes, you simply cannot get away from this."

The idea therefore arises that a hybrid approach may be necessary, or even a reconsideration of the work process itself, respondent E indicated: instead of building a fully automated system with covered logic and conditions, a simpler document-driven approach with free input fields might be more in line with practice. This shifts the responsibility for interpretation and testing back to the human practitioner, while retaining digital support where efficient.

5.4.2 Practice and Model

At first glance, the scheme does not seem to fit well in quadrant IV. The high complexity, the great social and financial importance, and the need for careful testing would tend towards a High/High approach. However, the low application frequency and the limited scalability of the scheme ensure that a high-performance custom application is hardly profitable (Ewers & Vessey, 1981). At the same time, the required development speed is relatively low.

The case thus shows that there is a tension between complexity and the value of automation. Where low-code in quadrant IV normally excels in simplicity and cost-effectiveness, in this case it appears that the complexity of the scheme exceeds the low-code platform. The need for simplicity in development processes clashes with the functional complexity of the scheme.

This confirms that the boundaries of the model are not absolute. On paper, the scheme functions within this quadrant due to few applications, low pressure and low standardization, but the substantive complexity pulls the case towards a hybrid form that falls outside the standard thinking of the model. In such cases, it is not only the nature of the scheme, but also the way in which the process is set up that determines the technological strategy. The use of low-code as a flexible shell on top of a simpler backend, or even a semi-manual workflow, can then be a suitable compromise. It is crucial that the technology adapts to the nature of the application and not the other way around.

5.5 Evaluation

The evaluation of the two-dimensional model based on the discussions with architects and project managers confirms that the proposed division into four quadrants fits in well with how technological choices are made in practice.

Within RVO, respondent M confirms the importance of a consistent technical foundation. In the hybrid approach used there, a conscious choice was made for standardization in the technical basis of the low-code platform, which allows new scheme applications to be realized more quickly because a lot of functionality is already 'given' in the form of preconfigured components. This approach ensures that low-code remains usable, but at the same time prevents the platform from becoming overloaded with functionality for which it is not intended.

Respondent M also points out that the maturity of the platform used is a determining factor in the experienced development speed, something that also influences the positioning within the model: an application in a low-pressure quadrant can still be realized efficiently due to higher platform maturity. At the same time, the conversation with respondents K and J shows that choices around pro-code versus low-code also depend on broader organizational considerations, such as the availability of modelers, the risk of vendor lock-in, and the extent to which a technology aligns with existing competencies. The choice for a low-code platform was

motivated by the desire to utilize the team's modeling capacity, but also the insight that there are major differences between platforms in terms of quality and suitability. Respondent L, who works in a pro-code-oriented environment outside the RVO, also emphasizes that technological choices are mainly sustainable when they are in line with the broader architecture of the organization. According to him, adding a hybrid or low-code solution to a primarily pro-code landscape often leads to conflicts in the areas of integration, management and architectural consistency. For example, the respondent indicates that low-code is only considered for small, standalone applications that do not require a connection to the primary system.

These insights underline that the two-dimensional model is a valuable framework for analyzing technological positioning, but that in practice it is also necessary to include architectural, organizational and strategic boundary conditions. The quadrants are recognizable and explanatory, but the precise positioning of an application is partly determined by factors such as platform maturity, development strategy and technical governance. Practice shows that the model is able to meaningfully indicate both standard choices and borderline cases. This makes the model robust as a reflection instrument, provided it is applied with an eye for context and architectural coherence.

6. Discussion

In this chapter, the research results are discussed based on the two central sub-questions. While chapter 5 describes the empirical findings from the case studies, this discussion focuses on the meaning of those findings in relation to the theoretical framework and practice. The interpretation is thus focused on the underlying mechanisms that explain why certain technology choices are effective within specific application contexts.

6.1 Categorization Using Efficiency Dimensions

The first sub-question of this research is: *How can applications be categorized?* This question is a crucial step in answering the main question, because a clear categorization is necessary to make well-considered choices between low-code, pro-code or a hybrid development strategy. In this study, a division was chosen based on two axes, namely 'need for development efficiency' and 'need for execution efficiency'. This division not only appears logical from a practical perspective, but is also supported theoretically.

The choice of 'need for development efficiency' as a dimension is based on the recurring importance of development speed, manageability and time-to-market within government organizations. In the literature on low-code, this dimension is repeatedly put forward as the most important reason for adoption (Alsaadi et al., 2021; Magesa & Jonathan, 2021). At the same time, the extent to which development is complex or dependent on integrations strongly determines the technological approach (Cabot, 2020). These factors come together in what is summarized in this research as development efficiency.

The second dimension, ‘need for execution efficiency’, focuses on the required quality and performance of an application during use. This includes scalability (Fernandes et al., 2020), simultaneous use (Pan et al., 2024) and the degree of automation (Ajimati et al., 2024). For public institutions, these are also critical success factors, because applications are often rolled out on a large scale under high usage pressure (McLean, 1979).

What makes this division strong is that the dimensions do not overlap but rather complement each other, creating a model that offers room for nuance and hybrid scenarios. The case analyses confirm the applicability of these dimensions. Respondents were able to position their applications within the model without much explanation and indicated that the trade-off between development speed and implementation requirements weighed in for their choice of technology. The model thus not only shows analytical value, but also practical usability. The categorization contributes to shared understanding between development teams, architects and policy makers, which is essential for the success of hybrid software development (Hoogsteen & Borgman, 2022).

However, it must be acknowledged that the boundary between development and execution aspects is diffuse in some cases. In applications with many policy-dependent exceptions or dynamic regulations, complexity and performance can be closely intertwined. This emphasizes that the model is indicative, but not a rigid classification scheme. Its strength lies in supporting professional considerations, not in prescribing fixed routes.

In summary, this research shows that categorizing applications based on development and execution efficiency is both theoretically sound and empirically useful. These dimensions make it possible to position applications in a systematic way and thus form a robust basis for answering the second sub-question about appropriate development strategies per category.

6.2 Matching Development Strategy to Application Categories

The second sub-question focuses on the translation of application categories into a suitable development strategy: *For each application category, what specific combination of low-code and pro-code design would enhance both efficiency and effectiveness in implementation?* Based on the positioning within the developed two-dimensional model, four quadrants are distinguished, as described in section 6.1. Each quadrant represents a specific application category that results from a combination of development and implementation

efficiency requirements, with associated implications for the use of low-code, pro-code or a hybrid approach. In the analysis below, these categories are discussed separately. To improve clarity and referential ease, each category is given a descriptive name that reflects their strategic nature.

The first category includes applications that require both rapid development and robust implementation. These applications typically combine time pressure with complex functional requirements. In this category, a mixed or hybrid development approach proves to be most effective. Standard functionalities and user interfaces can be realized quickly with low-code platforms, while pro-code remains essential for technically demanding components (Dong et al., 2024; Buist, 2024). This category, combining speed and precision, is referred to as Hybrid Precision. This term emphasizes a development context in which agility is necessary, without sacrificing quality or structural robustness. At the same time, hybrid development is also vulnerable, as technical issues quickly arise without clear governance or architectural frameworks (Pan et al., 2024). A successful application of the Hybrid Precision category therefore requires not only the right technological mix, but also organizational maturity in team structure, tooling, and governance (Hoogsteen & Borgman, 2022).

The second category concerns applications where speed of development is essential, while the requirements for performance, scalability and technical depth remain relatively limited. In these cases, low-code can be applied in its purest form. The absence of heavy implementation requirements enables short development cycles and increases the involvement of end users in the development process (McLean, 1979). This category is therefore referred to as Rapid Agility, a term that reflects both the urgency and the lightness of the development process.

In contrast, the third category include applications that require high execution efficiency, but where the pressure for rapid development is relatively low. In these contexts, control, scalability, and maintainability are prioritized over speed. Pro-code development remains the

preferred strategy here, due to its flexibility and deep control over the technical architecture (Mosquera et al., 2024). This category is referred to as Robust Engineering, a term that refers to the planned and structured nature of the development process that is necessary to ensure long-term operational reliability. The cases studied show that the assumed slowness of pro-code can be partly overcome by team continuity, modular architecture, and standardized development practices, indicated respondent G. This emphasizes that the effectiveness of a development strategy is often determined more by organizational processes than by the properties of the technology used.

Finally, the fourth category concerns applications with minimal requirements in both development and implementation. In these scenarios, low-code offers a very efficient solution, with minimal effort and low costs (Ewers & Vessey, 1981). This category is referred to as Minimal Investment, which emphasizes that both technical and organizational efforts can and should be kept to a minimum (Ajimati et al., 2024; Marek et al., 2021). At the same time, several cases show that complexity can also be hidden in this category. In seemingly simple processes, non-trivial implementation challenges can arise, for example in specialist policy domains. This confirms that the model provides direction, but does not constitute an absolute decision rule. Strategic technological decisions remain dependent on additional factors, such as interpretation needs, organizational context and platform capacity.

Together, these four categories, Hybrid Precision, Rapid Agility, Robust Engineering, and Minimal Investment, show that a strict distinction between low-code and pro-code does not do justice to the complexity of application development in governmental organizations. The choice of a technology is not a goal in itself, but a strategic means that must be aligned with both the technical characteristics of the application and the organizational context in which it is developed. The model contributes to this strategic alignment by providing insight into which factors determine the effectiveness of a development strategy. In doing so, it not only provides

an analytical framework, but also a practical tool for IT decision-making in complex governmental environments.

7. Conclusion, Implications, Limitations, and Future Research

This final chapter provides a comprehensive overview of the key findings of this research. First, the main conclusions are presented and the central research question is answered. Next, the theoretical and practical implications are discussed in detail. Finally, the limitations of this study are outlined, along with suggestions for future research to build on and refine the presented framework.

7.1 Conclusion

This research provides guidance on how to effectively combine low-code and pro-code development approaches to optimize both efficiency and effectiveness in the context of government application development. Driven by the rapid adoption of low-code technologies within governments and the simultaneous need for robust, scalable and secure systems, the research addresses an urgent need for guidance on hybrid software strategies.

The central research question is: *Which combined design of low-code and pro-code would optimize efficiency and effectiveness for implementing applications in a governmental environment?* To answer this question, a two-dimensional assessment model was developed, based on both theoretical literature and empirical insights gathered through ten semi-structured interviews with IT professionals. Applications were positioned along two axes, namely need for development efficiency and need for execution efficiency. This positioning results in four different quadrants, each representing a certain category of applications. Each of the categories is linked to a recommended development strategy: pure low-code, pure pro-code or a hybrid approach.

The main conclusion of this research is that no single development method is universally superior. The optimal approach depends on the specific characteristics of the application and the context. By using the different techniques in the right places, effectiveness can be ensured.

Low-code is particularly effective when time-to-market (Baumgarten et al., 2024) and simplicity in development are paramount and technical complexity is low (Carroll & Maher, 2023). Pro-code is preferred in situations that require maximum control, security and performance (Pan et al., 2024). Hybrid models, although more complex in terms of governance and technical coordination, allow organizations to benefit from the strengths of both development methods when development speed and performance need to be optimized together (Buist, 2024; Metrolho et al., 2019). The proposed two-dimensional model provides a practical, accessible tool for decision-making in IT projects within the government. By translating abstract efficiency and effectiveness criteria into concrete development guidelines, the framework enables consistent, justifiable and context-sensitive technology choices. Furthermore, the successful application of the model in practical cases at RVO demonstrates its practical relevance and applicability. The two-dimensional model is not a rigid classification, but a guiding framework that allows for considerations based on human interpretation, policy objectives and organizational capacity.

In conclusion, this research offers a scientifically sound and practically applicable model that bridges the gap between the development speed of low-code techniques and the certainty of pro-code. It advocates well-considered, context-driven decisions about software architecture and lays the foundation for a more integrated and adaptive approach to digital transformation in the public sector.

7.2 Implications

7.2.1 Theoretical Implications

This research contributes to the existing academic literature in the domain of information and software development, and in particular to the relatively young research field around low-code and hybrid development strategies, in several ways.

First, the research addresses a clearly identified knowledge gap regarding the role of low-code technologies in professional IT environments, especially when combined with traditional pro-code methods (Alsaadi et al., 2021; Carroll & Maher, 2023). By developing a two-dimensional assessment model consisting of the axes ‘need for development efficiency’ and ‘need for execution efficiency’, this research provides a theoretical framework with which applications can be systematically classified based on their technological requirements. This approach complements existing IT strategy models, such as the Strategic Impact Grid by Nolan and McFarlan (2005), but at the same time offers a more concrete application focused on technology choices within government organizations.

In addition, the model contributes to the conceptualization of hybrid software architectures. The four identified application categories, Hybrid Precision, Rapid Agility, Robust Engineering and Minimal Investment, not only provide practical tools, but also form a basis for further theoretical exploration of the conditions under which hybrid development strategies are successful. In contrast to limited binary comparisons between low-code and pro-code, this research positions technological choices as a spectrum, influenced by functional needs and environmental factors.

Finally, this research points to new directions for further theoretical research into the role of software development strategies within digital transformation. By clarifying the tensions between speed, scalability and complexity within software development, the research contributes to a more nuanced understanding of how organizations can navigate between standardization and customization in an increasingly dynamic technological environment. In doing so, the research offers a broader insight into how development strategies fit within changing forms of IT management and innovation, and forms a starting point for further theory development on technological decision-making in government contexts.

7.2.2 Practical Implications

In addition to the theoretical contribution, this research also offers clear implications for practice, in particular for IT managers, architects and policy makers within government organizations that are faced with the strategic choice between low-code, pro-code or a hybrid approach. The developed two-dimensional framework functions as a decision-making tool with which applications can be systematically assessed based on their need for development and implementation efficiency. This clarification of relevant assessment criteria supports managers in making substantiated choices that go beyond cost reduction or development speed alone, by including a total of eight aspects in the consideration.

The practical value of the model lies in improving alignment between IT and business. By using recognizable, quantifiable characteristics, the model promotes a shared conceptual framework that simplifies communication between content experts, development teams and management. This prevents misunderstandings and accelerates decision-making about technology deployment, in particular in multidisciplinary project environments in which different perspectives come together. The framework can be used for this purpose when drawing up the target architecture of the applications. In addition, the research underlines the importance of realistic expectations regarding development technologies. Although low-code can deliver fast results, it also entails risks in terms of vendor lock-in, maintainability and dependency on platform suppliers. Pro-code approaches, on the other hand, face challenges, such as longer development times and a greater dependency on specialized knowledge, but at the same time offer more opportunities for optimization. By increasing this awareness and translating it into concrete considerations using the model, organizations can better anticipate the long-term consequences of their technology choices.

In summary, this research enables managers to make technology decisions that are not only efficient and effective, but also sustainable, controllable and tailored to the specific context of governmental organizations.

7.3 Limitations and Future Research

While this research provides valuable insights into strategically combining low-code and pro-code in a government context, it has some limitations that should be considered when interpreting the results.

A first limitation concerns the context-specific nature of the empirical part. The case studies were conducted within one government organization, RVO, which strengthens the internal validity of the model, but limits its generalizability to other domains or sectors. Although an additional validation interview was conducted at an external government agency, the model has not yet been tested in other types of organizations, such as commercial companies, healthcare institutions or educational institutions, where different priorities and technological constraints may apply. Future research should therefore examine the model's applicability across a broader range of sectors, particularly those in which the trade-off between development speed and implementation quality takes on a different character. Such research could offer valuable insights into the model's robustness, adaptability, and the need for sector-specific refinements.

Second, the assessment model is based on qualitative measurements along two axes, development and implementation efficiency, with eight characteristics scored subjectively. Although these characteristics were carefully selected based on literature and practical experiences, their interpretation remains partly dependent on human judgment. This introduces a degree of variability in the application of the model, especially when the scores are determined by different stakeholders with different perspectives. Future research could focus on developing standardized measurement instruments or quantitative scales to further objectify the application of the model.

A third limitation concerns the dynamics of technological developments. The market for low-code platforms is very dynamic, with new functions, integration options and AI support being added at a rapid pace. This creates the risk that certain conclusions or strategic

recommendations will become less applicable as the platforms evolve. Regular updates of the model based on technological progress is therefore recommended.

Finally, an interesting point of attention for future research is the increasing importance of the Hybrid Precision category. In the current model, the boundaries of the quadrants are determined based on the center of the axes. As low-code platforms offer increasingly advanced hybrid options and thus also become suitable for applications with more demanding implementation requirements, the Hybrid Precision category may increase in size and relevance. Future research could therefore focus on refining the boundaries between the quadrants, for example by introducing adaptive or weighted boundary values. In addition, it is recommended to investigate longitudinally how these boundary shifts develop over time, so that the model can be made more resistant to technological changes and thus more future-proof.

In conclusion, the results of this research form a solid basis for further academic and practical deepening. At the same time, the limitations underline the importance of continuous evaluation and contextual adjustment when applying the developed model.

References

Ajimati, M. O., Carroll, N., & Maher, M. (2024). Adoption of Low-Code and No-Code Development: A systematic literature review and future research agenda. *Journal of Systems and Software*, 112300. https://doi.org/10.1016/j.jss.2024.112300

Alsaadi, H. A., Radain, D. T., Alzahrani, M. M., Alshammari, W. F., Alahmadi, D., & Fakieh, B. (2021). Factors that affect the utilization of low-code development platforms: survey study. *Romanian Journal of Information Technology and Automatic Control*, *31*(3), 123–140. https://doi.org/10.33436/v31i3y202110

Asawa, K., Kukreja, S., & Gondkar, R. (2021). An NCDP for developing a Blockchain based dynamic supply chain management with auto-generation of smart contract. *2022 27th International Conference on Automation and Computing (ICAC)*, 1–6. https://doi.org/10.23919/icac50006.2021.9594235

Baumgarten, C., Endl, R., & Stich, S. (2024). Professionelle Softwareentwicklung mit Low Code optimieren – eine Fallstudie. *HMD Praxis der Wirtschaftsinformatik*, *61*(5), 1213–1234. https://doi.org/10.1365/s40702-024-01095-y

Benbasat, I., & Vessey, I. (1980). Programmer and Analyst Time/Cost estimation. *MIS Quarterly*, *4*(2), 31. https://doi.org/10.2307/249335

Bexiga, M., Garbatov, S., & Seco, J. C. (2020). Closing the gap between designers and developers in a low code ecosystem. *MODELS '20: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, *61*, 1–10. https://doi.org/10.1145/3417990.3420195

Bhattacharyya, S. S., & Kumar, S. (2021). Study of deployment of "low code no code" applications toward improving digitization of supply chain management. *Journal of Science and Technology Policy Management*, *14*(2), 271–287. https://doi.org/10.1108/jstpm-06-2021-0084

Buist, M. (2022). *Barriers for empowering an agile workforce with Low-Code*. https://arno.uvt.nl/show.cgi?fid=167644

Cabot, J. (2020). Positioning of the low-code movement within the field of model-driven engineering. *MODELS '20: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, *76*, 1–3. https://doi.org/10.1145/3417990.3420210

Cabot, J., & Clariso, R. (2022). Low code for smart software development. *IEEE Software*, *40*(1), 89–93. https://doi.org/10.1109/ms.2022.3211352

Carroll, N., & Maher, M. (2023). How Shell fueled digital transformation by establishing DIY software development. *MIS Quarterly Executive*, 99–127. https://doi.org/10.17705/2msqe.00076

Carroll, N., Ó Móráin, L., Garrett, D., & Jamnadass, A. (2021). The importance of citizen development for digital transformation. *Cutter IT Journal*, *34*(3), 5–9.

Chrysler, E. (1978). Some basic determinants of computer programming productivity. *Communications of the ACM*, *21*(6), 472–483. https://doi.org/10.1145/359511.359523

Di Ruscio, D., Kolovos, D., De Lara, J., Pierantonio, A., Tisi, M., & Wimmer, M. (2022). Low-code development and model-driven engineering: Two sides of the same coin? *Software & Systems Modeling*, *21*(2), 437–446. https://doi.org/10.1007/s10270-021-00970-2

Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, *119*, 87–108. https://doi.org/10.1016/j.jss.2016.06.013

Dong, Y., Kong, L., Zhang, L., Wang, S., Liu, X., Liu, S., & Chen, M. (2024). A search-and-fill strategy to code generation for complex software requirements. *Information and Software Technology*, 107584. https://doi.org/10.1016/j.infsof.2024.107584

Dunsmore, H., & Gannon, J. (1979). Analysis of the effects of programming factors on programming effort. *Journal of Systems and Software*, *1*, 141–153. https://doi.org/10.1016/0164-1212(79)90014-1

Elshan, E., Dickhaut, E., & Ebel, P. A. (2023). An investigation of why low code platforms provide answers and new challenges. *Proceedings of the Annual Hawaii International Conference on System Sciences*. https://doi.org/10.24251/hicss.2023.746

Ewers, J., & Vessey, I. (1981). The systems Development dilemma - A programming perspective. *MIS Quarterly*, *5*(2), 33. https://doi.org/10.2307/249222

Fernandes, J. P., Araújo, R., & Zenha-Rela, M. (2020). Achieving scalability in project based learning through a Low-Code platform. *34th Brazilian Symposium on Software Engineering (SBES)*, 710–719. https://doi.org/10.1145/3422392.3422482

Funk, D. (2022). Creating a Low-Code business Process execution platform with Python, BPMN, and DMN. *IEEE Software*, *40*(1), 9–17. https://doi.org/10.1109/ms.2022.3212033

Harel, E. C., & McLean, E. R. (1985). The effects of using a nonprocedural computer language on programmer productivity. *MIS Quarterly*, *9*(2), 109. https://doi.org/10.2307/249112

Harris, L. R. (1984). Natural language front ends. In *The AI Business: Commercial Uses of Artificial Intelligence* (pp. 149–162). The MIT Press eBooks. https://doi.org/10.7551/mitpress/1165.003.0015

Hevner, N., March, N., Park, N., & Ram, N. (2004). Design science in Information Systems Research. *MIS Quarterly*, *28*(1), 75. https://doi.org/10.2307/25148625

Hitt, L. M., & Brynjolfsson, E. (1996). Productivity, business profitability, and consumer surplus: three different measures of information technology value. *MIS Quarterly*, *20*(2), 121. https://doi.org/10.2307/249475

Hoogsteen, D., & Borgman, H. (2022). Empower the workforce, empower the company? Citizen Development adoption. *Proceedings of the 55th Hawaii International Conference on System Sciences*, *7*, 4717–4726. https://doi.org/10.24251/hicss.2022.575

Hurlburt, G. (2021). Low-Code, No-Code, what's under the hood? *IT Professional*, *23*(6), 4–7. https://doi.org/10.1109/mitp.2021.3123415

Indamutsa, A., Di Ruscio, D., & Pierantonio, A. (2021). A Low-Code development environment to orchestrate model management services. In *IFIP advances in information and communication technology* (pp. 342–350). https://doi.org/10.1007/978-3-030-85874-2_36

IV-architectuurteam. (2023). MPS 2023: Actualisatie Multi-platformstrategie RVO IST situatie. In *RVO*.

Khalajzadeh, H., & Grundy, J. (2024). Accessibility of low-code approaches: A systematic literature review. *Information and Software Technology*, *177*, 107570. https://doi.org/10.1016/j.infsof.2024.107570

Legtenberg, S. D. G. (2024). *Adopting citizen development successfully with the distribution of roles and responsibilities*. https://arno.uvt.nl/show.cgi?fid=167604

Lientz, B. P., Swanson, E. B., & Tompkins, G. E. (1978). Characteristics of application software maintenance. *Communications of the ACM*, *21*(6), 466–471. https://doi.org/10.1145/359511.359522

Lourenco, H., Ferreira, C., & Seco, J. C. (2021). OSTRICH - a Type-Safe template language for Low-Code development. *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 216–226. https://doi.org/10.1109/models50736.2021.00030

Magesa, M. M., & Jonathan, J. (2021). Conceptualizing digital leadership characteristics for successful digital transformation: the case of Tanzania. *Information Technology for Development*, *28*(4), 777–796. https://doi.org/10.1080/02681102.2021.1991872

Marek, K., Śmiałek, M., Rybiński, K., Roszczyk, R., & Wdowiak, M. (2021). BalticLSC: Low-Code software development platform for large scale computations. *Computing and Informatics*, *40*(4), 734–753. https://doi.org/10.31577/cai_2021_4_734

McLean, E. R. (1979). End users as application developers. *MIS Quarterly*, *3*(4), 37. https://doi.org/10.2307/249047

Meador, C. L., & Mezger, R. A. (1984). Selecting an end user programming language for DSS development. *MIS Quarterly*, *8*(4), 267. https://doi.org/10.2307/249096

Metrolho, J., Araújo, R., Ribeiro, F., & Castela, N. (2019). An approach using a Low-Code platform for retraining professionals to ICT. *EDULEARN19 Proceedings*, 7200–7207. https://doi.org/10.21125/edulearn.2019.1719

Moskal, M. (2021). No-Code application development on the example of Logotec App Studio platform. *Informatyka, Automatyka, Pomiary W Gospodarce I Ochronie Środowiska*, *11*(1), 54–57. https://doi.org/10.35784/iapgos.2429

Mosquera, D., Ruiz, M., Pastor, O., & Spielberger, J. (2024). Understanding the Landscape of Software Modelling Assistants for MDSE tools: A Systematic Mapping. *Information and Software Technology*, *173*, 107492. https://doi.org/10.1016/j.infsof.2024.107492

Nolan, R., & McFarlan, F. W. (2005). Information Technology and the board of directors. *Harvard Business Review*, *83*(10), 96–106. https://pubmed.ncbi.nlm.nih.gov/16250628

OECD. (2021). *The E-Leaders Handbook on the Governance of Digital Government*. OECD Publishing. https://doi.org/10.1787/ac7f2531-en

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. *2022 IEEE Symposium on Security and Privacy (SP)*. https://doi.org/10.1109/sp.2018.00005

Op ’t Land, M., Krouwel, M. R., & Gort, S. (2021). Testing the concept of the RUN-Time Adaptive enterprise: Combining organization and IT agnostic enterprise models with organization implementation variables and low code technology. In *Lecture notes in business information processing* (Vol. 411, pp. 228–242). https://doi.org/10.1007/978-3-030-74196-9_13

Overeem, M., & Jansen, S. (2021). Proposing a framework for impact analysis for Low-Code development platforms. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 88–97. https://doi.org/10.1109/models-c53483.2021.00020

Pacheco, J., Garbatov, S., & Goulao, M. (2021). Improving collaboration efficiency between UX/UI designers and developers in a Low-Code platform. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 138–147. https://doi.org/10.1109/models-c53483.2021.00025

Pan, Y., Shao, X., & Lyu, C. (2024). Measuring code efficiency optimization capabilities with ACEOB. *Journal of Systems and Software*, *219*, 112250. https://doi.org/10.1016/j.jss.2024.112250

Pantelimon, S., Rogojanu, T., Braileanu, A., Stanciu, V., & Dobre, C. (2019). Towards a seamless integration of IoT devices with IoT platforms using a Low-Code approach. *IEEE 5th World Forum on Internet of Things (WF-IoT)*, 566–571. https://doi.org/10.1109/wf-iot.2019.8767313

Pichidtienthum, S., Pugsee, P., & Cooharojananone, N. (2021). Developing module generation for Odoo using concept of Low-Code development platform and automation systems. *2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*, 529–533. https://doi.org/10.1109/iciea52957.2021.9436754

Reilly, J. (2021, November 5). *How No-Code platforms can bring AI to small and midsize businesses*. Harvard Business Review. https://hbr.org/2021/11/how-no-code-platforms-can-bring-ai-to-small-and-midsize-businesses

Schenkenfelder, B., Salomon, C., Buchgeher, G., Schossleitner, R., & Kerl, C. (2023). The potential of Low-Code development in the manufacturing industry. *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–8. https://doi.org/10.1109/etfa54631.2023.10275503

Tanimoto, S. L. (2013). A perspective on the evolution of live programming. *2013 1st International Workshop on Live Programming (LIVE)*, 31–34. https://doi.org/10.1109/live.2013.6617346

Van Deenen, J. (2024, April 30). RVO: We komen sterker uit deze crisis. *blueriq*. https://www.blueriq.com/actueel/rvo-we-komen-sterker-uit-deze-crisis

Walls, J. G., Widmeyer, G. R., & Sawy, O. a. E. (1992). Building an Information System Design Theory for Vigilant EIS. *Information Systems Research*, *3*(1), 36–59. https://doi.org/10.1287/isre.3.1.36

Wang, S., & Wang, H. (2024). A teaching module of No-Code Business App development. *Journal of Information Systems Education*, *32*(1), 1–8. https://aisel.aisnet.org/jise/vol32/iss1/1

Wang, Y., Feng, Y., Zhang, M., & Sun, P. (2021). The necessity of low-code engineering for industrial software development: A case study and reflections. *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 415–420. https://doi.org/10.1109/issrew53611.2021.00112

Zmud, R. W. (1980). Management of large software development efforts. *MIS Quarterly*, *4*(2), 45. https://doi.org/10.2307/249336