ICB
Institut fur Informatik und Wirtschaftsinformatik

Ulrich Frank
Pierre Maier
Alexander Bock

Low Code Platforms: Promises, Concepts and Prospects

A Comparative Study of Ten Systems

70

ICB-RESEARCH REPORT

UNIVERSITÄT DUISBURG ESSEN
Open-Minded

ICB-Research Report No. 70
December 2021

Die Forschungsberichte des Instituts für Informatik und Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The ICB Research Reports comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Authors

Ulrich Frank

Pierre Maier

Alexander Bock

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
D-45141 Essen

ulrich.frank@uni-due.de

pierre.maier@uni-due.de

alexander.bock@uni-due.de

Contact:

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
45141 Essen

Tel.: 0201-183-4041
Fax: 0201-183-4011
Email: icb@uni-duisburg-essen.de

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)
DOI 10.17185/duepublico/75244

Abstract

In recent years, the catchword “low-code” has evolved into what can be seen as a major trend in software development platforms. A growing number of vendors respond to this trend by offering software development platforms that promise limited need for coding only and a tremendous boost in productivity. Both aspects have been the subject of intensive research over many years in areas such as domain-specific modeling languages, model-driven software development, or generative programming. Therefore, the obvious question is how "low code" platforms differ from such approaches and what specific performance features they offer. Since there is no unified definition of “low-code”, the only way to develop an elaborate understanding of what it is – and might be – is to analyze the actual use of the term. For obvious reasons, it is not promising in this respect to rely on marketing announcements made by vendors. Instead, it seems more appropriate to examine “low-code” platforms. This research report presents a study of 10 relevant platforms, capturing and assessing common characteristics as well as specific features of individual tools. The study is guided by a method that consists of a conceptual framework, which provides a uniform structure to describe and compare “low-code” platforms, and a process model that describes the sequence of steps.

List of Figures

List of Tables

Table of Contents

1 Introduction

The short history of software development has produced a remarkable variety of approaches that aim at improving development productivity and software quality. Given the importance of software development and the notorious lack of programmers, there was always a great need for such approaches. Among the most prominent methods and tools are rapid prototyping, computer-aided software engineering (CASE) tools, fourth-generation programming languages (4GL), and model-driven development. A plethora of vendors developed tools that compete for a share of the huge market, which led to an almost unmanageable range of commercial tools and reusable software artefacts. For a while, then, productivity growth fell off in interest. One reason for this may be that productivity tools became a kind of commodity; another reason may be the excessive attention given to agile approaches.

Nevertheless, the lack of programmers and the unsatisfactory productivity of software development remained a pressing problem, the relevance of which is still increasing with the challenges of the digital transformation. It forces companies to reorganize their value chains or even to change their business models, which in turn requires them to rebuild their processes and products in software. Therefore, more tools and methods to increase the efficiency of implementing business software are needed like never before.

1.1 The Proliferation of a New Trend

In the last couple of years, presumably fostered by the omnipresent need for software developers, a new trend has emerged that brings back the quest for productivity in professional software development to the center stage. Under the heading of *low-code,* a new kind of platforms is announced that aims at clearly widening the bottleneck in software development by enabling an unprecedented increase in software development productivity. The term "low-code platform" (LCP) designates a class of application development platforms that aim at boosting development productivity and clearly reduce the amount of coding. They focus mainly, though not exclusively, on the development of business applications that are aligned with business processes. In addition, LCPs are often also intended to enable non-programmers ("citizen developers") to develop and adapt application systems that fit their personal needs.

Often, but not necessarily, low-code products are also touted to incorporate elements of a variety of other classical and recent notions in systems development, such as, beside the already mentioned "citizen developer", "DevOps", "user experience", "business process management" (BPM), "robotic process automation" (RPA), "IT-business alignment", "artificial intelligence" (AI), "machine learning" (ML), "microservices", and "cloud native".

Presumably coined in 2014 by a market research company (Forrester 2014), the label has been adopted by a considerable number of software vendors. The trend is promoted by fulsome

promises made by vendors and market research firms. They regard remarkable gains in development productivity, e.g., "Enterprise low-code application platforms deliver high-productivity and multifunction capabilities across central, departmental and citizen IT functions." (Vincent, Natis, & et al., 2020), or "Some firms are turning to new, 'low-code' application platforms that accelerate app delivery by dramatically reducing the amount of hand-coding required." (Wyatt, 2018)

Other characterizations focus on the use of visual representations and the empowerment of laypersons: "When you can visually create new business applications with minimal hand-coding — when your developers can do more of greater value, faster — that's low-code." [1] "That's where the power of citizen development comes in—with no-code/low-code platforms, anyone can build applications without software expertise, significantly faster, and at a fraction of the cost."[2]

The enthusiasm that carries the trend is additionally fueled by impressive success stories, e.g.: "T-Mobile US Inc. developed a mobile app for employees to share their availability when about 80% of the stores were temporarily closed. The app, built within a matter of days in March by a team of business analysts at T-Mobile's consumer markets organization using Power Apps, lets field leaders determine staffing of stores that remain open." (Agam Shah, 2020).

Market research firms present staggering predictions concerning the volume of the market for LCPs, which has aroused the interest of investors. According to the Wall Street Journal, Forrester is predicting the market for LCPs to grow by 15% to USD 7.7 billion in 2021 (Agam Shah, 2020). As reported by a business analyst, Brandessence, another market research firm, values the LCP market at almost USD 13 Billion already in 2020 and estimates a volume of USD 65 Billion in 2027 (Wyatt, 2018).

The trend is not restricted to industry. Recently, it was discovered by academia where it generated a considerable response. In 2020, a workshop series dedicated to low-code started as part of the established *Models* conference.[3] It attracted the largest number of participants of all *Models* workshops.

Our brief presentation of key features of the "low-code" trend allows us to draw the following preliminary conclusions:

[1] https://www.ibm.com/uk-en/automation/low-code, accessed 10-18-2021

[2] https://www.pmi.org/citizen-developer, accessed 10-18-2021

[3] https://lowcode-workshop.github.io/, accessed 10-18-2021

LCPs address a problem of utmost relevance. The continuing digitization of business processes, personal routines, and social interaction is accompanied by the extensive penetration of organizations, public spaces and private households with software. Software is changing the world – and changing the world, or just coping with a changing world, does not only require using software, but developing and maintaining it. As a consequence, the ability to develop, adapt and deploy software quickly is a pivotal factor of many company's competitiveness.

High relevance also from academic perspective. Even though methods and tools to support the efficient design and implementation of software systems as well as their alignment have been on the research agenda for long, various serious challenges remain that provide excellent opportunities for research. That suggests to also account for possible innovations offered by LCPs and relate them to corresponding research findings.

Lack of clear and coherent conceptualization. Even a cursory glance at the market indicates a wide range of products now sold under the label LCP. The diversity of these products as well as corresponding characterizations by vendors and market research firms are compromised by marketing jargon and hidden agendas. The most comprehensive definition of LCP that we could find comes from Gartner: "An LCAP is an application platform that supports rapid application development, deployment, execution and management using declarative, high-level programming abstractions such as model-driven and metadata-based programming languages, and one-step deployments." (Vincent et al., 2020) While this definition may seem satisfactory at first, on closer inspection, it shows an obvious lack of selectivity. High-level abstractions are characteristic for most software development systems, especially for those that make use of conceptual models. Also, it remains unclear what "metadata-based programming languages" are or what is meant by "one-step deployment."

In the academic literature, a clear conception of low-code has yet to evolve. Often, low-code platforms are characterized along lines even more vague than our generic outline above. Worse, some authors uncritically integrate marketing promises into their definitions. For example, one author suggests: "The low-code platform is a set of tools for programmers and non-programmers. It enables quick generation and delivery of business applications with minimum effort to write in a coding language and requires the least possible effort for the installation and configuration of environments, and training and implementation." (Waszkowski, 2019, p. 376) Similar description are provided by (Chang Young-Hyun & Ko Chang-Bae, 2017, 2017; Ihirwe, Di Ruscio, Mazzini, Pierini, & Pierantonio, 2020; Sanchis, García-Perales, Fraile, & Poler, 2020). Accordingly, the analysis of literature on low-code produces the insight that a concise definition of LCP, that would allow discriminating tools that fall under that term from others, does not yet exist.

Need for decision support. With respect to their prospective benefits, and given the considerable marketing activities of vendors, it is likely that managers who are responsible for providing

organizations with software get aware of LCPs and need to consider the introduction of an LCP. The complexity and diversity of LCPs creates the need to support decisions related to the selection, assessment, introduction and management of these tools.

1.2 The Need for an Exploratory Study

Apparently, the low-code trend is, to some degree, fostered by an often careless and ambiguous use of the term. This is nothing unusual for trends in IT. One could even suspect that ambiguity is an important driver of trends, since it allows addressing a wider range of possible associations. The history of IT is not only packed with corresponding examples such as, to name recent examples only, “cloud”, “microservice”, or “edge computing”, it also comprises a large variety of trends that disappeared soon after they emerged. Against this background the question arises as to how academia should deal with trends in general, with the low-code trend in particular.

We do not think that research should account for every new trend. A trend that merely promotes just another fad will usually not deserve our attention. Trends that build on grandiloquent promises or even myth should be approached with great caution, that is, we should be reluctant embracing them without a critical assessment. In the case of low-code, it is at first hard to determine, whether and to what extent promises and prospects are appropriate. At the same time, the relevance of the subject for both, research and practice, recommends the analysis and assessment of the trend. Such an analysis requires a clarification of the term. Otherwise, the subject of a respective investigation could not be clearly delineated.

The search for a convincing conceptualization of LCPs may follow three principle approaches. First, one could distil a definition from existing definitions and descriptions. In contrast, the second approach would rather follow an inductive schema. Based on an analysis of platforms that carry the label LCP, one would search for common properties to condense them in a corresponding concept. Finally, one could take a more prescriptive approach and define the term in a consistent and purposeful way without paying much attention to actual uses of the term. While the last approach obviously corresponds to our responsibility to create and maintain elaborate and consistent terminologies, it would probably result in a concept of LCP that could be perceived as artificial and offensive, because it would likely exclude systems from carrying a label they are known for. With regard to the lack of convincing definitions, the first approach may be useful only with respect to the objective to develop a coherent concept that does not contradict the use of the term in practice. Therefore, the most appropriate way to develop a conceptualization of LCP or, at least, contribute to the clarification of the term, is the analysis of products that carry the label.

From a scientific perspective, the study of products is, for serious reasons, a critical issue. In general, scientific studies should focus on general knowledge about a certain subject that is

applicable to a wide range of specific cases. Therefore, research should primarily aim at concepts and theories rather than at the analysis of particular products. Apart from being specific, insights into a certain product are not only volatile, because a product may change, they may also be compromised by the fact that the subject, especially in the case of software products, is of limited accessibility. Available descriptions of software may be insufficient or misleading. Furthermore, the effort required for a comprehensive analysis of complex software systems will often be beyond the available resources.

We were therefore initially hesitant to carry out a product study. However, in the end, we decided to take this unusual step for two important reasons. First, as we explained already, the low-code trend concerns a challenge that is of outstanding economic relevance and relates to research topics at the core of business information systems. Second, the lack of a clear conceptualization implies the need for clarification. This lack of a comprehensible definition does not only compromise decisions related to the selection and use of LCPs, it also makes it widely impossible to include the subject in university curricula. Third, the study is exploratory, that is, the analysis and assessment of the selected tools is not its main purpose. Instead, the investigation of products serves the development of a conceptual foundation that goes beyond volatile product features and the identification of possible future research topics.

In addition to this report, our research on LCPs is presented in two further publications. An overview of the study that includes specific technical aspects of the tools is presented in (Bock & Frank, 2021a). (Bock & Frank, 2021b) aims in particular to clarify the concept of LCP.

1.3 Outline of the Investigation

The exploratory study aims at a clarification of the subject. To this end, we shall, at first, focus on the following questions.

1. What is an appropriate method for the analysis of LCPs?
2. What are criteria relevant for the selection of an LCP?
3. What images of LCPs do vendors create?
4. What are characteristic features of LCPs?
5. How well do they fulfil promises made by vendors?
6. How do LCPs compare with the current status of research, and what, if any, technological innovations are realized by these platforms?
7. What opportunities for future research arise from the present attention to low-code development?

Further questions may evolve during the course of the investigation.

To avoid possible pitfalls of a product-centric study, we will present a method that guides the investigation. At its core is a conceptual framework. It provides the concepts used to describe and analyze the selected products in order to avoid bias through the use of product-specific

terms. In addition, the method includes a process model that describes the course of the investigation. At its core, the study comprises an analysis of selected LCPs that follows a common scheme. It concludes with a discussion of the results with specific emphasis on opportunities for future research.

We start with a brief analysis of the subject addressed by LCPs, that is, of software development productivity and user empowerment. At first, we will look at principle enablers of software development productivity and related design conflicts. Against this background, we give an overview of research aimed at productivity and end-user programming. The examination of foundational aspects and related work serves two purposes. On the one hand, supports the development of the conceptual framework. On the other hand, it is a prerequisite for assessing whether and, if so, to what extend LCPs go beyond existing research results.

2 Background and Related Research

Ever since the emergence of software engineering, it has been a pivotal goal of the discipline to improve the efficiency of software development and maintenance. Software engineering research in pursuit of this goal has produced such well-known methods and tools as rapid prototyping, computer-aided software engineering (CASE) tools, fourth-generation programming languages (4GL), and model-driven development. Similarly, business information systems research has long directed its efforts at the economic realization and adaption of information systems. Among the best-known outcomes of corresponding research are methods for conceptual modeling, methods for enterprise modeling, and, of particular significance, reference models.

Before we give a structured overview of research that concerns promises made by LCPs, we briefly discuss principle objectives and challenges related to the economic construction of software.

2.1 Software Development Productivity: Inhibitors, Drivers, and Conflicts

With respect to both, promoting software development productivity and decreasing development costs, reuse of existing artefacts is of outstanding relevance. It requires the identification of existing or possible commonalities shared by a set of systems. In order for software artefacts to be reusable, they need to be integrated with other artefacts. Hence, *integration* is a further enabler of reuse. During their lifetime, most software systems need to be adapted to changing requirements. That recommends designing systems for adaptability in order to avoid excessive maintenance costs.

The *complexity* of many software systems is an obstacle to the efficiency of the software development process as well as to the quality of software, since it does not only increase the effort required for system development, but also the risk of failure. The development of application systems usually requires collaboration of different stakeholders such as users, system analysts, programmers, etc. The notorious cultural chasm between these groups is a source of friction and misunderstanding, which may seriously compromise the efficiency of the development process and the quality of its outcome. Furthermore, the design of business software often requires analysis and change of the organization of the action system to be supported by the software. Hence, there is need for mutual adaptation ("business-IT alignment"), which increases the overall complexity of the analysis and development process.

For promoting drivers and mitigating inhibitors of software development productivity, *abstraction* is of pivotal relevance. Table 1 provides a cursory representation of how abstraction effects the various aspects of software development productivity.

Table 1

Problem	Measure
Complexity/Risk	Reduction of complexity through *abstraction* from irrelevant aspects to gain a clearer view of relevant aspects.
Communication	Focus on concepts different stakeholders are familiar with through *abstraction* toward common concepts.
Adaptability	Accounting for possible and probable change through *abstraction* from properties that may change over time, especially from particular technologies.
Reuse	Identification of commonalities shared by different use contexts through *abstraction* toward common properties.
Integration	Enabling communication between software components through *abstraction* toward common concepts that, e.g., allow the specification of interfaces.
Organizational Integration ("Business-IT Alignment")	Enabling a clear correspondence between action system and software systems through *abstraction* toward common concepts to avoid friction.

Table 1: The pivotal role of abstraction

In order for an abstraction to become effective in a software development process, it has to be captured in some kind of *linguistic representation*. On the one hand, that is the case for a representation in implementation languages, for example, by expressing a generalization with a superclass. On the other hand, there is also need for representing an abstraction in a language the various stakeholder involved in a software development process are familiar with, e.g., by stating "A bachelor thesis is a thesis." As a consequence, there is need to bridge the semantic gap between the universe of discourse (represented by the language spoken in a particular domain) and the required implementation language(s).

In other words: there is need for *conceptual models*. First, they enable representations of software systems by using concepts domain experts are familiar with and that can be transformed to implementation documents, such as code. Second, they support communication between different stakeholders by focusing on concepts and, thus, by fading out implementation level peculiarities. In addition, *enterprise models* support organizational integration of software systems by integrating conceptual models of software with conceptual models of the corresponding action system, such as business process models.

To systematically reduce the complexity of a system, it is useful to focus on certain aspects of a system, and fade out others. One common approach to achieve this kind of reduction is to focus on one of three generic dimensions of a software system (and of action systems as well). *Static* abstractions focus on data. A typical example would be a data model. *Functional* abstractions serve representing the functions offered by a system and, maybe, data exchanged between functions. Finally, *dynamic* abstractions serve capturing the dynamics of a system, that

is, possible changes of its state or processes. While abstracting on one specific dimension helps with reducing complexity, the design of a software systems requires the integration of corresponding models, e.g., of data flow diagrams and data models. Therefore, the languages and tools used to create these models should support their convenient and consistent integration.

Abstractions in general, conceptual models in particular, are suited to promote reuse and, hence, to increase productivity. However, with respect to the purpose of our investigation, we need to take a closer look at the effects of abstraction on reuse. First, the *productivity* that results from reusing an artefact depends on how well it fits a specific requirement. In other words, the more specific an artefact is, the higher is its potential contribution to increasing productivity. For example, if one wants to develop a planning tool for sales representatives, a reusable model that was designed exactly for this purpose will promote productivity significantly more than a general model of planning tasks or even a generic computing model like a spreadsheet. However, at the same time, the costs required for the acquisition of a more specific model will usually be clearly higher than those of a more general model that allows for a clearly wider *range of reuse* or, put differently: for much better economies of scale. To sum up this obvious conflict succinctly: semantics of a reusable artefact promotes reuse productivity, but is an obstacle to the range of reuse.

A similar conflict arises for integration. The integration of two artefacts implies common (interface) concepts (such as, e.g., data types or classes). The more specific these common concepts are, that is, the more semantics they represent, the higher is the degree of integration. A high degree of integration translates into effective and consistent communication between the connected artefacts. If, for example, two systems share a common concept of invoice, exchanging a particular invoice will be more efficient and less error-prone than representing invoices as strings only. However, this kind of selectivity, as with any communication, has its downside, too: it is an obstacle to openness, because it excludes all artefacts that do not share the specific concept. To mitigate this problem, it is popular to recommend “loose coupling”, that is, interfaces with little specific semantics, only. Usually, the downside of this recommendation, i.e., the resulting lack of efficiency and integrity, remains unmentioned. With respect to its cause, this conflict is similar to the previous one. Semantics represented by an interface promotes efficiency and integrity of integration, but is a threat to openness, cf. (Frank, 2011b).

The adaptability of an artefact depends on its semantics, too. The more generic an artefact is the wider is the range of particular solutions it can be adapted to. A spreadsheet program is an illustrative example of this effect. At the same time, it restricts the range of possible adaptations clearly less. Hence, semantics limits adaptability, but fosters the integrity of customizing an artefact. At best, an artefact is based on abstractions that cover the range of conceivable changes, but do not allow for other changes. That, of course, requires a reliable theory of the domain(s) targeted by an abstraction.

All these conflicts relate to a further conflict that concerns the ease of (re-) using artefacts to build custom solutions. The more the representation of an artefact (its structure, its functions and its behavior) corresponds to concepts a prospective user is familiar with, the less is the effort the user has to put into learning the use of the artefact. For example, hiding technical terms like “file” or “data type” from a user with no programming skills, and instead presenting her with concepts like account, customer etc., will likely increase her ability and willingness to make use of the corresponding artefact. At the same time, it is likely to reduce the range of problems, the user can address.

The brief overview of abstraction and the challenges concerning its proper application serves two purposes. First, it allows a characterization of the research approaches presented in the next section. Second, it allows an assessment of the platforms analyzed in the study with respect to the trade-offs they offer to cope with those principal design conflicts. From a vendor’s perspective, this is a critical issue with respect to the market segments that should be covered by a particular artefact and to economies of scale that might be achieved.

2.2 Models and Languages

Different from other kinds of models, conceptual models are created with a specific modeling language. The following approaches aim at promoting the productivity of model-based software development.

2.2.1 Reference Models

While conceptual models promise considerable benefits, their construction from scratch is a serious obstacle to their use. *Reference models* represent a convincing approach to address this challenge (cf., e.g., Becker & Delfmann, 2007; Fettke & Loos, 2007; Frank, 2007). A reference model aims to represent not only one specific system, but an entire class of systems. To this end, reference models come with both, a descriptive and a prescriptive claim. First, the design of reference models aims at the representation of commonalities shared by a range of systems. Second, reference models, at the same time, go beyond existing systems and their peculiarities by aiming at improving the current state.

In an ideal case, they are developed with outstanding expertise and great care. Hence, they promise to improve the quality of software and to decrease development costs at the same time. These tantalizing prospects face two serious obstacles. First, it is clearly more expensive to develop a reference model than a particular model of a specific system only. Therefore, it requires convincing incentives for first movers to invest into the development of a reference model. Often, corresponding incentives are missing, because both effort and return on investment are hard to predict in advance (Frank & Strecker, 2007). Second, reference models need to be adapted to specific requirements. The adaptability of reference models to specific re-

quirements depends on the abstractions they feature. In the case of data or object models, generalization is an attractive option, because it enables specialization as monotonic extensions that do not compromise the reference model. However, specialization allows for limited adaptation only. Various more specific approaches exist that aim at more versatile adaptation (see, e.g., Becker, Delfmann, & Knackstedt, 2007).

Process models do not allow for specialization (Frank, 2012), because inserting further activities into a process model does not represent a monotonic extension. As a consequence, a key advantage of generalization does not come into play: whenever a general process is modified, the effects of this modification on a specialized process are not clear. For a thorough discussion of this restriction see (Frank, 2012). Various approaches to express abstractions over a range of specific process models have been proposed, such as "behavioral profiles" (Smirnov, Weidlich, & Mendling, 2007) or "families" of process variants (Milani, Dumas, Ahmed, & Matulevičius, 2016).

2.2.2 Domain-Specific Modeling Languages

Apart from controlled adaptation, any modification is conceivable that can be expressed in the underlying modeling language – at the risk of jeopardizing the integrity of a model. General-purpose modeling languages (GPMLs) like the UML or the ERM are still widely used, but come with a serious restriction. They require the design of models from scratch with basic concepts like *Class*, *Attribute* etc. As a consequence, the construction of large models is not only cumbersome, but also error-prone. Different from GPMLs, *domain-specific modeling languages* (DSMLs) provide concepts that represent a reconstruction of technical terminologies used in certain domains. Thus, they free designers from defining domain-concepts themselves. In addition, they support the construction of consistent models, because they restrict the scope of possible models, or systems respectively, to those that a DSML allows to express. Last but not least, DSMLs support the use of domain-specific notations, which foster comprehensibility of models.

Research has produced a large variety of DSMLs (cf., Frank, 2011a; Frank & Bock, 2020; Völter, 2013) and various tools that support the convenient implementation of model editors for given DSMLs (see, e.g., Frank, 2016; Gulden & Frank, 2010; Kelly & Tolvanen, 2008). A combination of reference models and DSMLs represents an especially powerful foundation for reuse and adaptation. First, a DSML reduces the effort required to develop a reference model. Second, it fosters comprehensibility and, thus, the reusability of models.

The design of a DSML needs to account for the conflict between range of reuse and productivity of reuse. While a very specific DSML, e.g., one to model products of a specific car vendor, promises to boost modeling productivity, its range of reuse is more limited than that of a more general product configuration language, which in turn enables more favorable economies of

scale. The only way to mitigate this conflict is to allow for additional abstraction (see, e.g., Frank, 2014).

2.2.3 Model-Driven Software Development

Usually, but not necessarily, conceptual models are not executable. Also, for good reasons, they abstract specific implementation details such as peculiarities of the user interface or of distribution, away. Therefore, it is required to transform models into executable representations such as code and to add missing aspects to the implementation. This does not only require considerable effort, it is error-prone, too.

Research on model-driven software engineering (MDE) (cf., e.g., Brambilla, Cabot, Wimmer, & Baresi, 2017; France & Rumpe, 2007) "… is primarily concerned with reducing the gap between problem and software implementation domains through the use of technologies that support systematic transformation of problem-level abstractions to software implementations. The complexity of bridging the gap is tackled through the use of models that describe complex systems at multiple levels of abstraction and from a variety of perspectives, and through automated support for transforming and analyzing models. In the MDE vision of software development, models are the primary artifacts of development and developers rely on computer-based technologies to transform models to running systems." (France & Rumpe, 2007, p. 37).

MDE is not only concerned with the development of software, but with supporting the entire software lifecycle. Corresponding tools cover a wide, not clearly specified range, from model editors, model checkers and model generators. Since models and code are usually represented in separate documents, various approaches aim at relaxing the problem of synchronizing models and code, usually by total or incremental re-generating code from modified models (cf., e.g., Massoni, Gheyi, & Borba, 2011; Razavi, Kontogiannis, Brealey, & Nigul, 2009). A more versatile approach is built on a common representation of models and programs, which allows for the execution of models and the modification of a system using the representation of choice (T. Clark, Sammut, & Willans, 2008; Frank, 2014).

In addition to research on model-driven software development, the Object Management Group (OMG) aimed at developing a standard architecture for model-driven development, called MDA ("model-driven architecture"). Among other things, it aims at promoting the reuse of models and at protecting investments into models (for an overview, see contributions in (Paige, Hartman, & Rensink, 2009)). To that end, it provides for avoiding dependencies from implementation languages and platforms. Since conceptual models abstract certain aspects away, these have to be added (see above). Instead of representing these aspects, like GUI elements, with constructs determined by a specific technology, MDA proposes to use a platform and implementation language independent model (PIM) to represent these additional system properties. Hence, a PIM can be used for a wide range of specific platforms, which requires a transformation of a PIM into a platform-specific model.

2.3 Reusable and Adaptable Software Artefacts

Software artefacts designed for reuse may be domain-independent such as operating systems, database management systems, GUI libraries, middleware systems, code generators, compilers, etc., or domain-specific. In contrast to conceptual models, reusable software artefacts do not require transformation into code. Nevertheless, to satisfy specific requirements they need to provide some kind of adaptation mechanism. Similar to the design of DSMLs, the construction of reusable artefacts has to cope with the conflict between range and productivity of reuse. From a vendor's perspective, this is a critical issue with respect to the market segments that should be covered by a particular artefact. In addition to representing commonalities of a range of systems, reuse also requires concepts that allow for convenient and safe adaptation to individual requirements. Convenient adaptation depends crucially on the concepts models are based on, and on how these are represented to the (re-) user.

2.3.1 Frameworks

The idea of frameworks has its origins in the realization that often a large number of similar systems exists that were developed independently of each other. Similar to a reference model, but on the level of already implemented software, a framework represents reusable and, hopefully, invariant commonalities of a range of systems to clearly decrease production and maintenance costs. A framework is an incomplete software system that is designed for adaptability (Codenie, Hondt, Steyaert, & Vercammen, 1997; Fayad & Johnson, 2000).

Frameworks provide for two principle kinds of adaptation. Black box reuse is restricted to specific interfaces that allow for extensions, e.g. so called "plugins", which do not affect the core of the framework. White box reuse allows for adapting code, ideally through specialization and overriding existing methods. While white box reuse enables a clearly higher degree of adaptability, it is more demanding and creates the problem that the introduction of new versions of a framework requires costly adaptations.

Frameworks comprise infrastructure and development systems like persistency frameworks that hide the peculiarities of actual persistency technologies from application systems or the *Eclipse Modeling Framework* (EMF) (Steinberg, Budinsky, Paternostro, & Merks, 2009) that serves the realization of modeling tools and code generators. Similar to most persistency frameworks, EMF is aimed at professional software developers. Domain-specific frameworks represent a common core of a range of domain-specific application systems. Prominent examples of domain-specific frameworks include skeleton enterprise systems for the financial industry (Bohrer, 1998), or frameworks that can be customized to individual web shops. Domain-specific frameworks, too, are aimed at professional developers. Making them accessible to non-programmers would require representations that abstract implementation related issues widely away (see 2.4). Also, supplementing frameworks with corresponding conceptual models is beneficial in that respect.

2.3.2 Components and Services

In addition to frameworks that emphasize a top-down approach, reuse can also be promoted in a bottom-up fashion. This kind of reuse directly corresponds to the idea of building software from pre-fabricated components of high quality, a vision that has inspired software engineering from its very beginning (Naur & Randell, 1969). Regardless of its charm, this vision is fraught with significant challenges that relate to the selection and composition of reusable software artefacts.

Various approaches have evolved that support finding and selecting reusable software artefacts such as components or services (which provide access to components or, in general, to software while abstracting the implementation away). The idea of structured annotations of components that enable effective automated search procedures has been around for long (Prieto-Díaz, 1991). It evolved into approaches to allow for richer descriptions of services in general, for web-services in particular (Garriga & Flores, 2019; Virgilio & Bianchini, 2010). These approaches are based on representations of service descriptions on a conceptual, rather than on a pure technical level. The performance of search procedures can be further improved by the use of thesauri that extend the search space to service descriptions by accounting for related terms, too.

After a software artefact that seems to fit a certain purpose was identified, it needs to be integrated with other software artefacts in order to realize a complete software system. At first, these approaches focused on "weaving" components into application systems (e.g., Schneider, 1999). Later, the focus shifted towards the composition of software systems through services. The idea of developing enterprise software by first designing business processes and subsequently automate them by selecting "best of breed" services, became especially popular. Corresponding research produced numerous proposals to efficient and consistent service composition (cf., e.g., Moulin & Sbodio, 2005; Tilsner, Fiech, Zhan, & Specht, 2011), in part with the intention to support domain experts with selecting and composing services (e.g., Weber, Paik, & Benatallah, 2013). Recent work on "on-the-fly computing" follows this path. It aims at enabling the automated composition of customized services from services offered somewhere in the Internet (Karl, Kundisch, Meyer auf der Heide, Friedhelm, & Wehrheim, 2020).

In any case, bottom-up building blocks need to be supplemented with knowledge, provided, for example, by reference architectures or models, in order to guide their composition to integrated systems.

2.4 End-User Empowerment

Ever since the invention of the personal computer there have been numerous research approaches aimed at making the solution space opened up by software accessible to those who

do not know the art of programming. On the one hand, that would help to address the notorious shortage of programmers. On the other hand, it would be a contribution to the empowerment of users by enabling them to develop software that fits their needs and by relaxing their dependency from professional developers.

To reach this goal, users need to be presented with representations that they are able to understand without massive training. There are two principle approaches to achieve this goal. On the one hand, simplified computer models aim at opening the power of the machine (or at least a notable part of it) to users by hiding confusing peculiarities of general-purpose programming languages, such as the dichotomy of types and instances, data and event management, or specific aspects of GUI implementation. On the other hand, domain-oriented representations abstract the computer even further away and offer representations that relate directly to concepts of the domain the user is familiar with.

2.4.1 Focus on Simplified Computer Models

Probably the most prominent example of simplified computer models are spreadsheet programs. The first spreadsheet-like program was available on mainframe computers in the late sixties of last century. However, only with *VisiCalc*, the first implementation for personal computers, the triumphal march of this software began. Even though spreadsheet programs did not result from a specific stream of research in computer science or software engineering, but from product-oriented research, they triggered various research approaches. Some focus on improving the integrity of spreadsheets by adding type information (Mendes et al., 2017) or by tools that allow detecting inconsistencies (e.g., Ahmad, Antoniu, Goldwater, & Krishnamurthi, 2003; Dou, Cheung, & Wei, 2014). Others aim at improving spreadsheet quality by providing specific analysis and design methods (Hermans, Pinzger, & van Deursen, 2011; Mendes et al., 2017).

SQL may be regarded as a further example, because it enables access to a database through set-oriented expressions, the use of which does not require specific programming skills. For those users who nevertheless feel overwhelmed by learning SQL, "query-by-example" (Zloof, 1975) is intended to provide a more intuitive access to a database. A user is presented with a table that she fills with values characteristic of the data objects sought. While such an approach does not empower users to create databases on their own, it allows convenient access to a database, provided the intended search is not too complex.

A further approach of this kind is "programming by example" (St. Amant, Lieberman, Potter, & Zettlemoyer, 2000) which is aimed at inductively generating programs from a set of examples, which may consist of input-output pairs or examples of intended behavior.

Visual programming (Costagliola, Deufemia, & Polese, 2004; Ingalls, Wallace, Chow, Ludolph, & Doyle, 1988), too, aims at abstracting the peculiarities of program code away. To this end, visual programming languages provide icons that represent basic programming constructs.

The icons can be annotated and arranged to represent a certain control flow. Among other things, they are used for the implementation of simple workflows, user interfaces or database queries.

2.4.2 Focus on Domain-Oriented Representations

The second principal approach, which may be combined with the first one, aims at providing users with concepts that correspond directly to the technical terminology they are familiar with, and with a functionality that that is tailored to a specific class of domain-specific tasks. Apart from domain-specific application systems that allow for user-driven adaptation, the most important concretization of this principal approach are the already mentioned domain-specific languages, which may be represented in various forms such as textual, graphical, or through tables. Examples include DSMLs that enable engineers to specify control software for devices such as refrigerators (Völter, 2013, pp. 18ff.) or watches (Kärnä, Tolvanen, & Kelly, 2009).

The field of “end-user computing” or “end-user development” (Nardi, 1995), which aims at "empowering end-users to develop and adapt systems themselves" (Lieberman, Paternò, Klann, & Wulf, 2006, p. 1), comprises various streams of research that aim at supporting users with developing small programs. While they also include approaches mentioned in the previous section, they emphasize the use of domain-specific concepts. Similar to model-driven development (MDD), “model-based development” follows the idea to generate code from models. However, different from MDD, it does not address professional developers, but users with no programming skills. A user “just provides a conceptual description of the intended activity to be supported and the system generates the corresponding interactive application” (Lieberman et al., 2006, p. 4).

3 A Method for the Analysis of the Low-Code Phenomenon

When we started to plan the study, it quickly became clear to us that the peculiarities of the object of study required a specific method. First, the analysis of complex software systems is fundamentally associated with specific challenges. They relate to the availability and reliability of relevant information, the availability of the platforms, and the effort required for a differentiated analysis. Second, the identification of the research subject itself is confronted with an idiosyncratic problem. The fact that there is no clear conceptualization of LCPs represents the pivotal motivation for conducting an exploratory study. However, without such a conceptualization the selection of platforms is a delicate task.

The outline of the method we used in the study follows a common concept. That is, a method comprises two main components. First, it includes some kind of linguistic structure that serves to purposefully represent the subject of an investigation (subchapter 3.1). Examples of respective linguistic structures include technical terminologies, conceptual frameworks, or, in the case of a modeling method, modeling languages. Second, it includes a process model that guides the course of an investigation (subchapter 3.2). To that end, the process model refers to the linguistic structure.

3.1 Conceptual Framework

Our analysis distinguishes between two facets of the low-code phenomenon. First, and most important, we aim at the identification and investigation of characteristic features offered by LCPs, which includes an assessment of their contribution to development productivity and to empowerment of non-professional developers. The impact of the phenomenon on the practice of software development in organizations is likely to also depend on the image of LCPs vendors created to market their products. The following framework addresses both facets.

3.1.1 Focus on the Analysis of Low-Code Platforms

In the previous chapter we gave on overview of research that is aimed at improving software development productivity and at user empowerment. The semantic net shown in Figure 1 shows a high-level representation of relevant concepts.

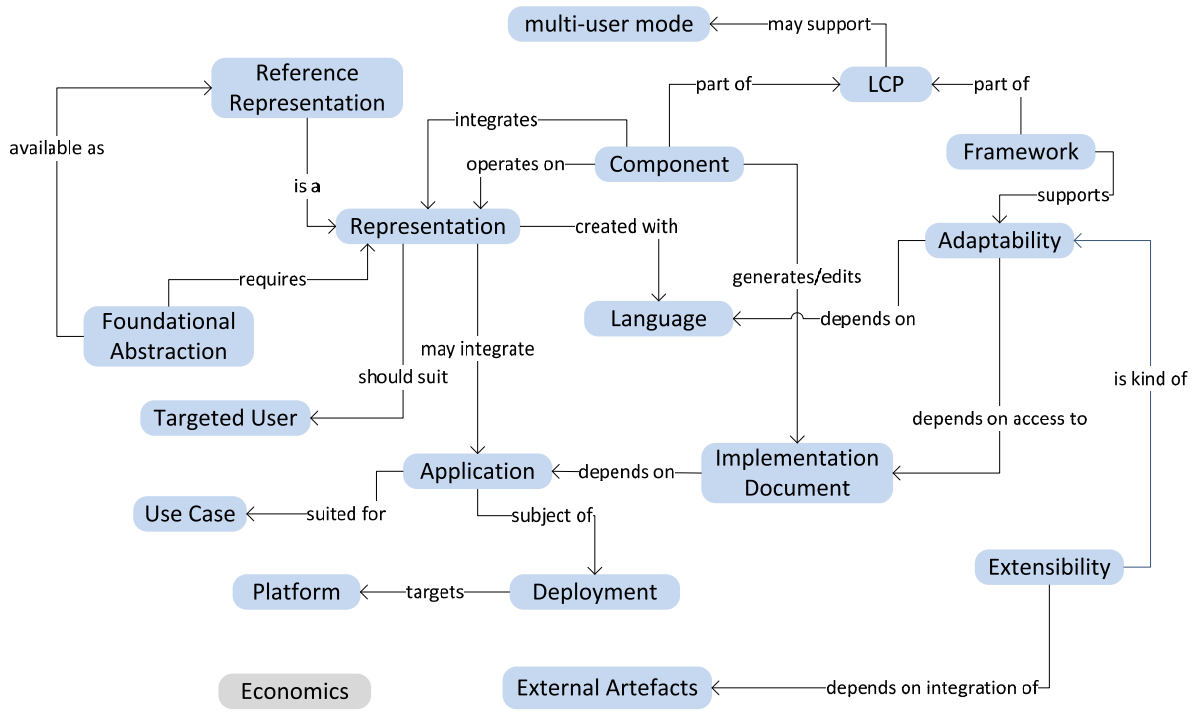

Figure 1: Semantic net of core concepts related to productivity and user empowerment

Foundational abstractions focus on static, functional, or dynamic aspects of a system. Conceptual models are especially suited to represent abstractions for the purpose of software development. But other (conceptual) representations like tables are also conceivable. Reuse is promoted by reference representations such as reference models or reference data structures. The languages used to create conceptual models can be either generic or domain-specific. The latter promote modeling productivity, but may restrict the bandwidth of use cases that can be covered.

A further kind of abstraction concerns the interaction between a system and its users. Corresponding models often represent interactive user interface elements (“widgets”) and their layout within a reference frame, e.g., a window or a frame.

In addition, software systems can be subject of reuse. These may comprise (software) frameworks or exemplary applications. In any case, reusable artefacts can be characterized by their specificity, ranging from generic artefacts like a persistency framework for storing objects in a relational database to domain-specific artefacts like a reference data model for a certain industry.

Reuse demands for adaptability. Adaptability of a representation depends on the abstraction enabled by the language the representation is created with. If, for example, a data modeling language allows for generalization, specialization could be used for safe, that is monotonic, and convenient adaptations. Especially in cases where representations do not cover all features of a system, adaptability also depends on the ability to access and modify implementation

documents such as code or schemas. Furthermore, a system can be adapted by extending it with external artefacts such as components or services.

With respect to characterize an LCP in comparison to other kinds of SDEs, the range of applications, or use cases, it is suited to cover, is an important criterion. Is it suited for any kind of application, be it small or large, local or distributed, or is it especially tailored for certain use cases?

Among the components of an LCP, those that operate on representations are especially relevant. Components like model editors support users with creating and managing representations. Furthermore, they may promote productivity by generating implementation documents. User empowerment depends chiefly on the representations users work on. The more they abstract specific implementation-related aspects away, the better they should be suited for laypersons. At the same time, professional developers may perceive it as a serious obstacle if implementation details are faded out.

The development of applications may require more than one developer. In that case, support for defining roles with specific access rights is a useful feature. At the same time, multi-user access may also be a feature required by applications created with an LCP.

Deployment can be a serious obstacle to the provision of applications. Especially, if multiple platforms have to be covered and new version have to be accounted for on a regular base. One approach to reduce deployment effort is to abstract peculiarities of platforms away by providing a runtime environment that is available for multiple platforms. This approach can be combined with the widely automated installation of software in data centers. Specific technologies to support automated scaling of applications such as containers and orchestrators provide additional support for deployment and maintenance.

Economics are of pivotal relevance for motivating the use of LCPs. It comprises acquisition costs, license fees, cost savings realized through productivity gains, training costs, as well as maintenance and deployment costs. In addition, economics of an LCP also depend on how well corresponding investments are protected. This relates especially to the portability of applications to other environments. Despite their relevance, we will address economic aspects only to a small degree, since acquisition costs and license fees were not available for all systems. While we aimed to shed light on the LCPs' effect on development productivity, a comprehensive assessment of this effect is beyond the scope of this study since possible productivity gains depends on multiple, partly contingent factors all of which could not possibly be accounted for.

In the following tables, the core concepts for the study are displayed. Table 2 represents criteria relevant for capturing the extent and quality of static abstractions enabled by a LCP. The ex-

tension of the static perspective through functions is captured in table 3. Subsequently, dynamic abstractions (table 4), GUI development (table 5), and further aspects for the evaluation of LCPs (table 6) are laid out.

Table 2: Concepts to characterize the static perspective

<table>
<tr><td colspan="3">Focus on Representations</td></tr>
<tr><td>Conceptual representations</td><td>Languages</td><td>Components</td></tr>
<tr><td>• data types
• data model
• object model
• …</td><td>• GPML (e.g., entity relationship model)
• DSML
• proprietary language</td><td>• model editor
• generator
○ database (DB) schema (DB type)
○ code (programming language)</td></tr>
<tr><td colspan="3">Focus on Reuse and Adaptability</td></tr>
<tr><td>Reference abstractions</td><td>Language concepts</td><td>Access to external sources and to implementation level documents</td></tr>
<tr><td>• generic abstractions
• domain-specific abstractions
Comment: represented by one or more conceptual representations listed in the leftmost column above</td><td>• generalization
• encapsulation
• composition
• …</td><td>• data models
• databases
• web
• files
• code
Comment: access to external sources will usually require interfaces such as ODBC, SQL, HTTP etc.</td></tr>
<tr><td colspan="3">Focus on Integration</td></tr>
<tr><td>Common static abstractions across a range of applications</td><td colspan="2">Comment: If two applications can (re-) use the same static abstraction, e.g., the same data model, the exchange of data between these applications is facilitated</td></tr>
<tr><td>Access to common data repositories across a range of applications</td><td colspan="2">Comment: If, in addition, both applications have access to a common data repository, e.g., a common database, instance-level integration (avoidance of data redundancy) is possible.</td></tr>
</table>

Table 3: Concepts to characterize the functional perspective

Focus on Representations		
Conceptual representations	*Languages*	*Components*

• function hierarchies
• data flows
• message flows
• business rules
• object models
• …
	• DFDs
	• UML
	• proprietary
	• DSML, e.g., rule specification language
		• model editor
		• generator
		  ○ code (programming language)
		  ○ interface definitions
		Comment: generators require access to corresponding static representations.

Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic abstractions • domain-specific abstractions ○ specificity *Comment:* represented by one or more conceptual representations listed in the leftmost column above	• composition • …	• (web) services • function libraries • class libraries • code • … *Comment*: access to external sources will usually require interfaces such as APIs and specific protocols like REST or SOAP

Table 4: Concepts to guide the analysis of dynamic abstractions

Focus on Representations		
Conceptual representations	*Languages*	*Components*

• business process models
• generic process models
• …
	• BPMN
	• UML
	• Petri net
	• state chart
	• proprietary
		• model editor
		• generator
		  ○ code (programming language)
		  ○ workflow schema
		Comment: generators require access to corresponding static and functional representations.

Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic abstractions • domain-specific abstractions *Comment:* represented by one or more conceptual representations listed in the leftmost column above	• composition • … *Comment*: reuse of processes is confronted with serious obstacles	• workflow schema • code (programming language) • … *Comment*: the execution of workflows may require access to external functions/services.

Table 5: Concepts to analyze support for GUI development

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*

• GUI models
• Generic process models
• …
	• GUI Editor
	  ○ drag-and-drop
	  ○ WYSIWYG
	  ○ platform-specific style
	  ○ code generation
		approach to integrate GUI with rest of application, e.g., model-view-controller (MVC) pattern

Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• library of general-purpose and domain-specific GUI elements (frames, widgets, ..) • example applications with reusable GUIs	• Customization of widgets • (re-) definition of styles and templates Adaptation to platform-specific look and feel	

Table 6: Further aspects for the analysis of LCPs

<table>
<tr><td colspan="2">Accessibility and Convenience of Use</td></tr>
<tr><td colspan="2">• guidance
  ○ support for methodical development
  ○ Built-in help function
• LCP user interface
  ○ inadequacies of LCP user interface
  ○ adaptable to different user groups
• modeling languages
  ○ disseminated
  ○ complexity, expressiveness</td></tr>
<tr><td colspan="2">Modes of Use</td></tr>
<tr><td>Platform (i.e., Application Development)</td><td>Application</td></tr>
<tr><td>• definition and management of access rights
• collaboration support (e.g., branching)
• use in distributed environments
• use in heterogeneous environments</td><td>• definition and management of access rights
• use in distributed environments
• use in heterogeneous environments
• collaboration support (e.g., collaborative editing)</td></tr>
<tr><td colspan="2">Focus on Deployment and Scalability</td></tr>
<tr><td>Target Location</td><td>Support</td></tr>
<tr><td>• local machine
• data center (cloud)
• supported platforms</td><td>• automated, transparent installation
• … across various platforms
• accessibility of deployed application (e.g., provided APIs)
• further tool support (e.g., containers, Kubernetes)</td></tr>
<tr><td colspan="2">Focus on Artificial Intelligence</td></tr>
<tr><td colspan="2">• internal AI services
• support for integration of external AI services</td></tr>
<tr><td colspan="2">Focus on Training and Support</td></tr>
<tr><td colspan="2">• Tutorials (manuals, screencasts, etc.)
• (online) courses
• external service providers</td></tr>
<tr><td colspan="2">Focus on Economics (criteria printed in grey not covered)</td></tr>
<tr><td colspan="2">• costs of use/ownership (acquisition, license fees, etc.)
• availability of trained developers
• contribution to development productivity
• protection of investment</td></tr>
</table>

3.1.2 Focus on History and Context

The second part of the framework aims at supporting a more comprehensive appreciation of the low-code trend by guiding an analysis of the relevant background, that is, the history of an LCP, its vendor's profile, and the image at which the vendor's marketing activities are directed.

Table 7: Provenance and image of low-code vendor

Profile of Vendor
Product Portfolio and Market Position
• range of development tools • relevance of LCP within product portfolio • range of further software systems • share of market
Product Provenance
• first labeled as LCP • platform predecessors • product evolution
Focus on Marketing
• unique selling proposition • targeted user groups • advertised use cases • promises

The framework provides a structure for analysis. It is primarily aimed at identifying those features that are characteristic for LCPs, and features that vary between the products included in the study.

3.2 Process Model

The high-level process model shown in Figure 2 gives an overview of how the method suggests to proceed with the present study. Note that the process in part reflects serious constraints we had to account for, such as limited time and resources.

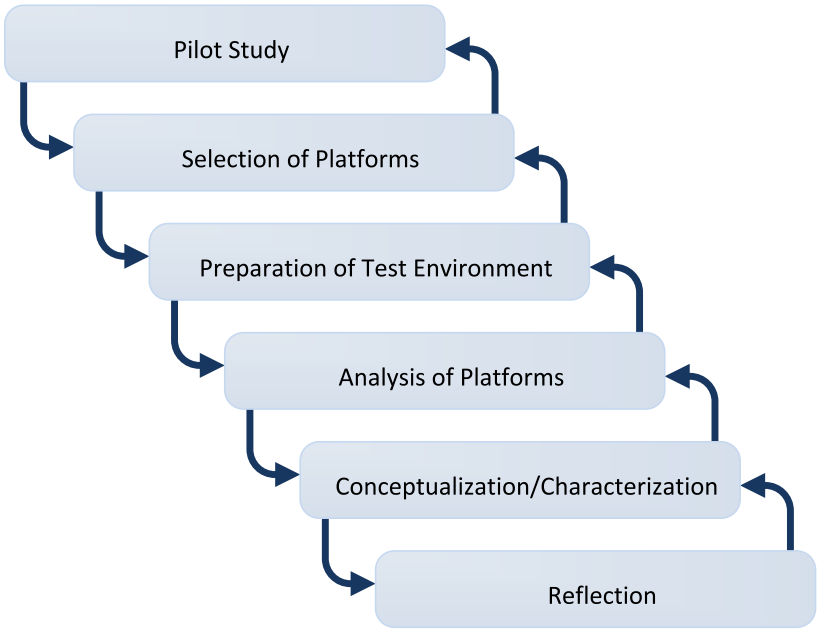

Figure 2: High-level process model of analysis method

Pilot Study. It is a characteristic property of most process models that are part of a method to start with a definition of the scope and the objectives of a project. This requires a delimitation of the object of study. In the case of our study, this request is both mandatory and beside the point. It is mandatory, because every investigation needs to focus on a certain subject. However, since a clarification of the subject is a main objective of the project, a clear demarcation of the research subject does not exist at first. The pilot study aims at relaxing this paradoxical situation by developing a preliminary notion of LCP that is suited to guide the necessary selection of platforms.

The pilot study starts with identifying a number of systems that are advertised as LCP. To that end, an iterative search of the web is combined with exploiting reports of market research firms and further publications on LCPs. For each LCP, characteristic features are recorded from the available sources. They comprise information on technical properties, on intended use and target user groups, as well as on specific characteristics of vendors, like size and product portfolio.

This analysis aims at developing a first idea of LCPs, which includes the identification of common properties and possible categories of LCPs in in the event that the systems under consideration differ significantly in some aspects. It cannot be the goal of this phase to create a representative selection of LCPs, since at this point too little is known about the systems. Nevertheless, the preliminary study should aim at accounting for platforms in every preliminary category that was identified. The process of searching for LCPs should be continued until a selection is achieved that is satisfactory in this respect. It needs to account, however, for project-specific constraints such as time and resources.

Selection of Platforms. Each platform that was identified in the previous phase needs to be reviewed as to whether it is suited for a closer inspection. Corresponding criteria include availability of products free of charge (this is not a mandatory request, but will often apply in research projects) and manageable installation effort. With respect to limited resources, it will often be required to limit the number of systems to analyze. To develop a rough idea of the effort, a platform that is likely to be included in the selection should be installed and analyzed with respect to the criteria proposed in the conceptual framework. Finally, this phase should produce the list of selected LCPs together with a rationale that is suited to justify the selection at this point in time.

Preparation of Test Environment. From a technical perspective, the test environment requires the installation of all selected LCPs. That may comprise the installation of or connection to additional components like DBMS or other systems in a tool chain. Each installation should be briefly documented, highlighting specific restrictions that may apply to the test version. In most cases, however, a comprehensive description of these restrictions is not possible. Alongside the *Analysis of Platforms*, this step also applies to each considered LCP separately.

Analysis of Platforms. This is the main phase of the study. It starts with a training phase to achieve a satisfactory degree of familiarity. For this purpose, it is a good idea to use existing training material, such as handbooks, screencasts, or examples provided with a platform. Note that this training phase is not explicitly documented in this report. Each platform and its vendor is then characterized according to the criteria provided with the conceptual framework. If the analysis is compromised by the lack of information or unreasonable effort, this is remarked. This is the case, too, for specific peculiarities that are relevant for the assessment of a platform, but are accounted for in the conceptual framework.

The analysis of each LCP follows three successive steps. First, the history and profile of the low-code vendor is explored following the structure from table 7. Second, the technological capabilities of the platform are examined according to the conceptual framework laid out in section 3.1.1. Third, a concise conclusion is presented that summarizes key aspects and reflects on the provision of abstractions, role of IT professionals, and scope of feasible applications within the platform.

Conceptualization/Characterization. Based on the descriptions of the selected platforms, this phase serves a final conceptualization of LCPs. That includes a critical review of the conceptual framework and, if required, a revision of the prototypical categories of LCPs deduced as part of the pilot study. If the diversity of platforms denoted as LCP does not allow for a precise conceptualization, an overview of commonalities and differences discovered within the range of selected systems is suited to represent the actual use of the term.

Reflection. The final phase serves two main purposes. First, it should produce a critical review of the study and its specific limitations. Second, it aims at assessing LCPs from a scientific

perspective. This recommends above all a comparison against the state of the art of related research. In addition, it could be analyzed whether and how the momentum created by the low-code trend could serve as an inspiration for future research.

The results from our *Pilot Study* and the *Selection of Platforms* (chapter 4) therefore precedes the analysis of respective LCP solutions in chapter 5. The *Conceptualization/Characterization* as well as a corresponding *Reflection* of the platform analyses is presented afterwards in chapter 6.

4 Selection of Platforms

Before conducting our in-depth analysis of low-code platforms, a list of potential candidate platforms needs to identified and selected. The first identification and selection of LCPs was achieved through a pre-study of available products and is elaborated in this chapter. Next to a derivation of possible categories of LCPs, a more comprehensive list of criteria is presented that serves to make a reasonable selection of platforms for our subsequent analysis.

4.1 Prototypical Categories of Low-Code Platforms

The first identification of LCPs was achieved by numerous means. At first, we examined reports from market research companies dating back to 2014 – the year the term "low-code" was apparently coined by Forrester (see chapter 1). At second, we researched product offerings by large and well-known companies such as Google, Salesforce, or Oracle. Some of the platforms identified in this second step might have been unnoticed in a mere exploration of market research companies' reports since, e.g., they might not meet the criteria of the market research company or might simply be too novel as to be considered in even the most recent reports. At last, we included platforms that we knew of for other reasons (e.g., platforms which might be relevant for our general research activity) that adopted the notion "low-code" in their marketing image. This first identification of platform products resulted in approximately 30 potential candidates for further in-depth analysis. Each of the identified LCPs was then subject to an exploratory survey, based on first impressions gathered during a preliminary look at the platforms, from which four prototypical categories were derived. These prototypical categories are (1) basic data management platforms, (2) workflow management systems, (3) extended, GUI- and data-centric integrated development environments (IDEs), and (4) multi-use platforms for business application configuration, integration, and development. The boundaries between these categories are fluid and, therefore, categories may overlap. Hence, the prototypical categories serve mainly to distinguish LCPs with respect to certain features that we consider particularly relevant with respect to our research goal.

Basic data management platforms resemble data management systems with an accompanying GUI design module. Within these LCPs, applications mainly serve to view and edit a limited range of data, organized in GUIs according to a user's needs. Productivity increase of these platforms shall be realized through providing a user-friendly management of entities. Further features for workflow management or functional specification might be partially considered but are not in the foreground. Platforms of the category *workflow management systems* focus the visual design of workflows with additional support for workflow execution and third-party connections. Opposed to basic data management platforms, the focus lies especially on the realization of dynamic aspects of an application. The availability of source code editors within these LCPs may vary, but is generally no necessary feature for using the platform. This is not

the case for *extended, GUI- and data-centric IDEs*. LCPs that are assigned to this prototypical category resemble regular IDEs either with extensive support to write and adjust source code within the platform or specific support to integrate source code files from external sources. Any efficient in-depth use of such platforms would thus demand some familiarity with programming languages. Typically, platforms assigned to this category do not have a specific focus on either data or workflow management. At last, *multi-use platforms for business application configuration, integration, and development* aggregate the preceding elements with an apparent focus on integrating and developing a variety of application artifacts. Like workflow management systems, these platforms emphasize the need to integrate several internal and external development artifacts.

The four prototypical categories serve as a first step towards structuring this opaque notion of "low-code" and associated products. At the same time, they served us as an orientation for a balanced, not necessarily a representative, selection of LCPs (see Section 4.2).

4.2 Selection Criteria

Given the four prototypical categories and a list of some 30 potential LCP candidates, it is required to argue for selection of some subset of LCP candidates that should be considered within our in-depth study. Given our research aim and analysis method, explained in chapters 1 and 3 respectively, we can note two straightforward aspects that must be fulfilled. The first aspect is the explicit use of low-code label by the vendor itself. This requirement must be met since we aim to conduct an inductive clarification of the notion "low-code" and do not desire to take over superficial features elsewhere associated with the label. The second aspect is concerned with accessibility. An elaborate, in-depth analysis of LCPs cannot rely on marketing material provided by vendors. Instead, it requires access to platforms. The consideration of these two criteria eliminates approximately 10 potential LCP vendors which include, among others, Salesforce, Google, Oracle, and ServiceNow.

The remaining LCP vendors were then classified according to criteria such as vendor size, market influence, intended user audience, market research companies' classification of the platform, and prototypical platform category. Two requirements served to further narrow down the selection. First, each of the identified prototypical categories shall be sufficiently represented, meaning that at least two LCPs from each category are to be considered. Second, the final selection focused on those remaining products that seem to have reached a considerable market position. They have been around for some time and are relatively well known. The use of this criterion is based on the assumption that most organizations will prefer platforms which are already established. The selection process resulted in the platforms presented in table 8. Included in this overview are also the most recent classification from Gartner and Forrester as of November 2021.

Table 8: Overview of selected low-code platforms

Section	Platform Name	Vendor	2021 Forrester Classification[4]	2021 Gartner Classification[5]
5.1 Basic Data Management Platforms				
5.1.1	Quickbase	Quickbase	n.a.	niche player
5.1.2	TrackVia	TrackVia	n.a.	n.a.
5.2 Workflow Management Systems				
5.2.1	Bonita Studio	Bonitasoft	n.a.	n.a.
5.2.2	Creatio Studio	Creatio	n.a.	niche player
5.3 Extended, GUI-, and data-centric IDEs				
5.3.1	Mendix Studio (Pro)	Mendix	leader	leader
5.3.2	WaveMaker Studio	WaveMaker	challenger	n.a.
5.3.3	Zoho Creator	Zoho	n.a.	n.a.
5.4 Multi-Use Platforms for Business Application Configuration, Integration, and Development				
5.4.1	Microsoft Power Apps	Microsoft	leader	leader
5.4.2	Appian	Appian	strong performer	challenger
5.4.3	Pega Platform	Pegasystems	strong performer	challenger

[4] https://reprints2.forrester.com/#/assets/2/54/RES161668/report, accessed 09-27-2021

[5] https://www.gartner.com/doc/reprints?id=1-27IIPKYV&ct=210923&st=sb, accessed 09-27-2021

5 Analysis of Platforms

The following inspection of the selected platforms is structured according to the preliminary categories identified in the pre-study (see chapter 4). Note that these categories may have to be revised and potentially adjusted following the study.

5.1 Basic Data Management Platforms

We consider "low-code" basic data management platforms to put a clear emphasis on features typically found in database management systems, such as the definition of entity types, data modeling, or system-managed persistence of data. Further aspects that are potentially relevant for application development might also be included.

5.1.1 Quickbase

The first versions of the platform Quickbase appeared about 20 years ago, making it one of the tools with the longest history. The vendor, also named Quickbase, has its headquarters in Boston, MA.

5.1.1.1 Quickbase: Profile of Vendor

Product Portfolio. The Quickbase platform is the only product offered by Quickbase.

Product Provenance. According to Quickbase, the company emerged in 1999 and adopted its current name in 2000.[6] An early marketing statement describes the Quickbase platform as a "web-based service […] to easily design and use databases" (Dec 11, 2001). Further assertions suggest that the early focus of Quickbase's marketing was to offer a platform as an improvement over mailing spreadsheets to support the management of "business information" for "everyday business people" (Jun 09, 2002). Application building is explicitly advertised since 2003 along the slogan "Build Solutions in Minutes, *Not Months*" (Sep 28, 2003). It cannot be clearly deduced from the archived web presence of the company what functionalities such an "application" is supposed to overcome. Around 2010, the platform is advertised as an "online database" (Jan 09, 2010) with an apparent focus on database application development. Prebuilt applications and "ready-to-use templates" (ibid.) are also mentioned – later in connection with the slogan "no code, compromise" (Apr 13, 2015). On the same date, support for "workflows" (solely for integrating data from external sources) and "team collaboration" is advertised. The Quickbase platform label "online database" was dropped on Dec 01, 2015. "Low & No Code Development" is mentioned explicitly since at least May 27, 2017 and the label "low-code platform" can be identified as early as Feb 12, 2018

[6] https://www.quickbase.com/about-us, accessed 08-04-2021

Focus on Marketing. Quickbase refers to its platform as low-code and no-code interchangeably. On the company's web pages, low-code is advertised as "a modern agile way to build and continually improve business applications" and a "visual app building approach"[7] with "pre-built templates and components that significantly reduce the amount of hand-coding needed to build."[8] RAD platforms, application platform as a service, or high-productivity application development platforms are said to be alternative notions for "low-code".[9] "No-code" is understood as a feature of an LCP.[10] According to Quickbase, low-code shall provide a higher degree of security, a more holistic approach to software development, a web-based UI, alignment of business and IT through overcoming the "barrier between organizations and their big ideas", e.g., by enabling "citizen development".[11] Use cases are related to, e.g., HR, sales or field data.[12] There is no focus on any specific domain.

Table 9: Quickbase profile of vendor summary

Product Portfolio
• low-code platform is the only product of Quickbase
Product Provenance
• designated as "low-code platform" since early 2018 • previously marketed as "online database" for "business people"
Focus on Marketing

• emphasis on "agility" and reusability
• low-code as a "modern agile way"
• "citizen development", platform for layper-sons
• promises
  ○ improved business-it-alignment
  ○ higher degree of "security"

[7] https://www.quickbase.com/business-application-platform/what-is-low-code, accessed 08-04-2021

[8] https://prod2.marketing.quickbase.com/resources/articles/low-code-development-platforms, accessed 08-04-2021

[9] https://prod2.marketing.quickbase.com/resources/articles/low-code-development-platforms, accessed 08-04-2021

[10] https://www.quickbase.com/business-application-platform/what-is-low-code, accessed 08-04-2021

[11] https://www.quickbase.com/blog/a-brief-history-of-low-code-development-platforms, accessed 08-04-2021

[12] https://www.quickbase.com/solutions-overview, accessed 08-04-2021

5.1.1.2 Quickbase: Analysis of Platform Features

The home screen of the cloud-based platform (see figure 3) consists of the available tabs, i.e., “My Apps”, “Pipelines”, and any additionally opened app, and an overview of the user’s applications. Within QuickBase, an “application” encompasses a user-specified set of tables as designed through a relational data model. Each application has exactly one data model associated with it and vice versa. Per default, each application is divided into “pages” – one for each user-specified table, one “Home” page, and one “Users” page (see figure 4). The pages that display tables (and, as such, exactly one entity type) allow for viewing, editing, and adding entities.

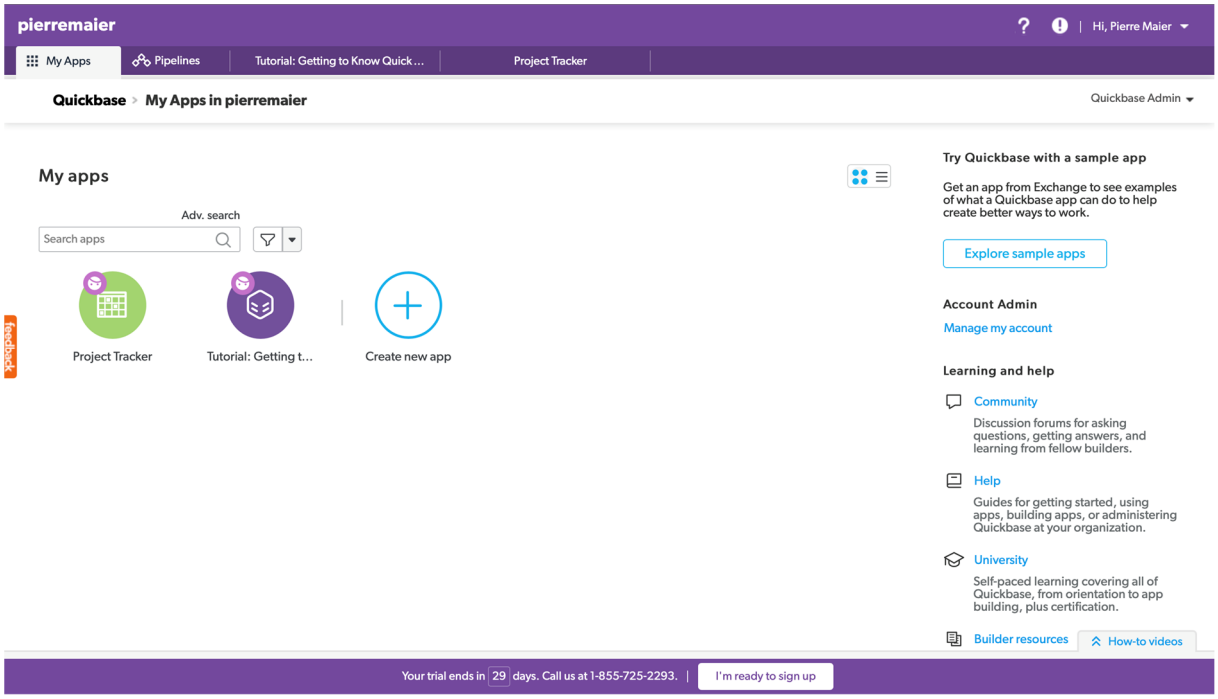

Figure 3: Quickbase “My Apps” overview screen

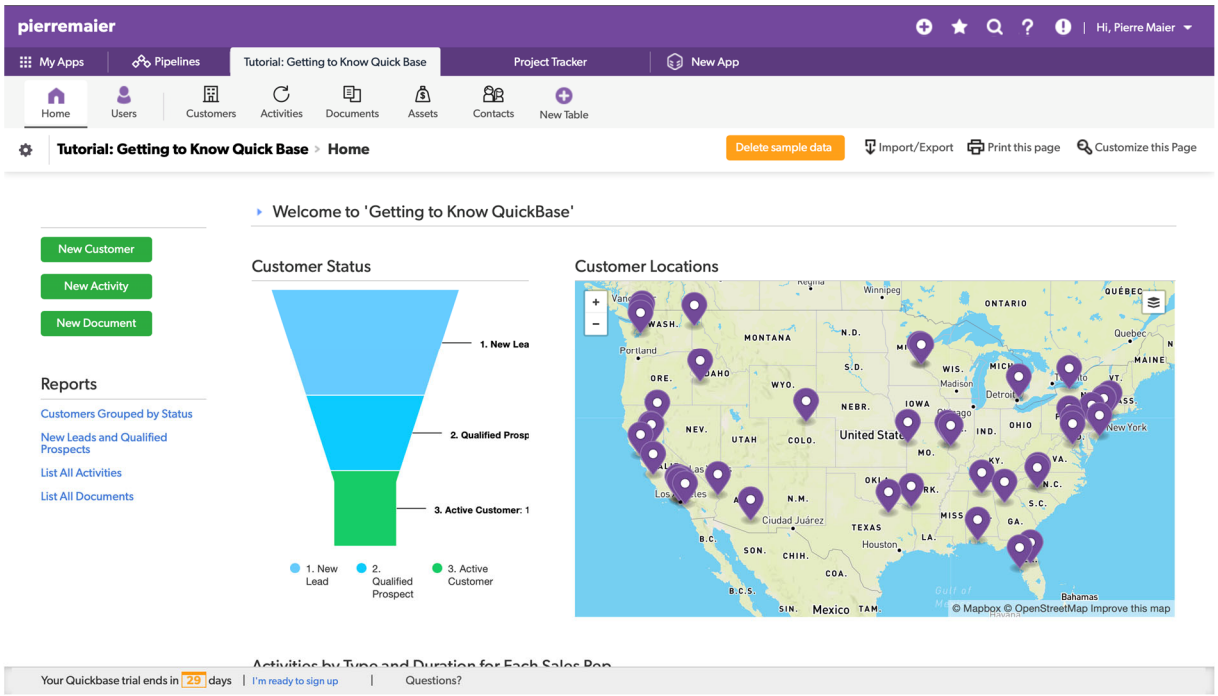

Figure 4: Quickbase example application homepage

Static Perspective. The core of each Quickbase application is a data model (exemplified in figure 5). The data model editor is called the "Visual Builder". In the "Visual Builder", entity types and associations between them can be defined. Next to familiar data types like string ("text") or integer ("numeric"), some generic business-oriented data types like "address" and "phone" are available. The values of attributes can be calculated through "formulas", which are similar to common spreadsheet functions. It is not possible to add user-specified data types. Furthermore, a user is restricted to using one-to-many relations in the Visual Builder. The missing possibility to model one-to-one relations poses a clear threat to system integrity. To model many-to-many relations in the Visual Builder an auxiliary entity type must be defined, a typical workaround for relational databases. The data that is used by Quickbase applications is persisted by the platform, the exact mechanisms and internal database schemata are inaccessible, omitting any further assessment in this regard. Each application relies on its own user-specified data model. Cross-references to data models from other (Quickbase or external) applications are not supported. Data reference models might be available as part of entire applications that can be acquired through the *Quickbase Exchange* app library. Otherwise, it is possible to select one of five data model "templates", which are small data models consisting of approximately three to seven interrelated entity types that can be adjusted by the platform user according to one's needs (see figure 7).

Table 10: Quickbase static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• data model	• largely inaccessible proprietary language	• diagram editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• general data types for strings, numbers, etc. • some generic business data types like “address” or “phone” • data model “templates” available and adaptable	• one-to-many relations between entity types	• no access to implementation-level documents • persistence on Quickbase platform
Focus on Integration		
Common static abstractions across a range of applications	Data models cannot be shared across Quickbase applications.	
Access to common data repositories across a range of applications	Data model of each application is managed separately. No common data repository can be access by Quickbase users.	

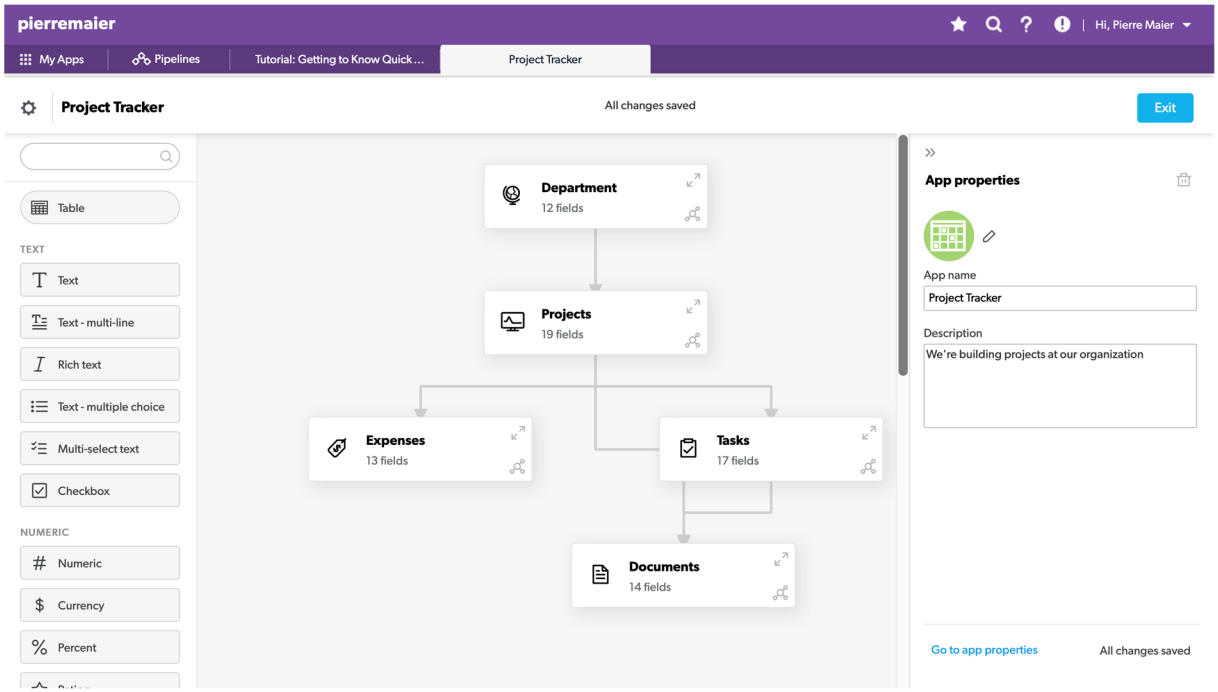

Figure 5: Quickbase visual data model example

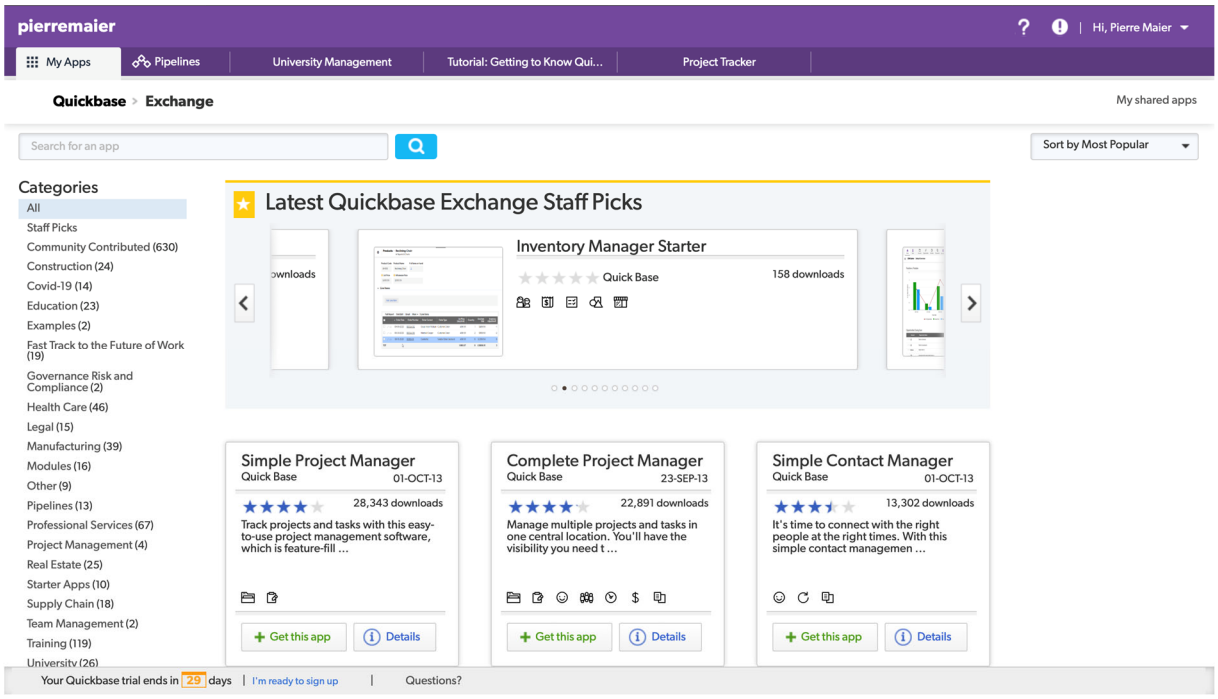

Figure 6: Quickbase Exchange

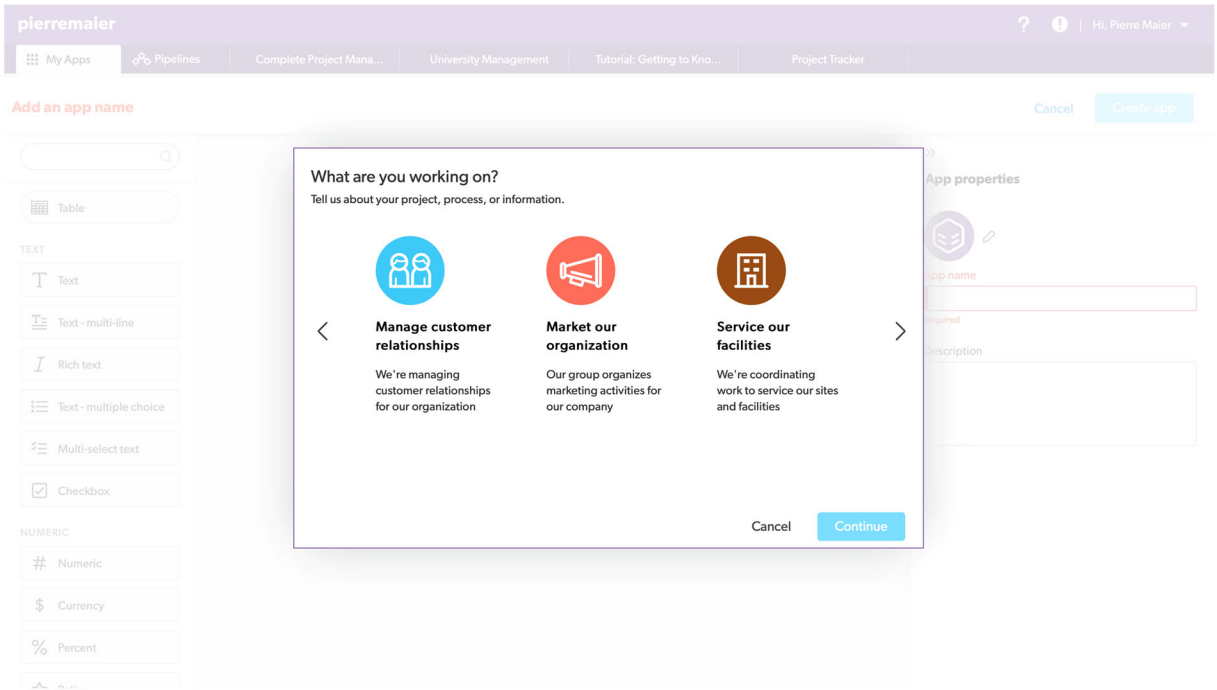

Figure 7: Quickbase application templates

Functional Perspective. Regarding user-specified functions in Quickbase, three mechanisms can be distinguished. First, it is possible to define the computing of attribute values through "formulas" (see *Static Perspective*). Second, users of Quickbase can specify so-called "Quickbase actions". Quickbase actions consist of a CRUD (create, read, update, delete) operation and an

additional condition, e.g., to remove all customers from the customer table, whose last purchase is older than five years. CRUD operations as well as conditions are system-defined and only the required variables (i.e., the attributes) and values must be provided by the user. Quickbase actions can only be accessed within the same Quickbase application. Third, so-called "channels", themselves part of Quickbase "pipelines" (see *Dynamic Perspective*), allow to specify functions for use across multiple Quickbase applications. Within pipelines, a channel can be used as a "trigger", a "query", or an "action" (see figure 8). External functions can be accessed via the pre-defined "action" channels in Quickbase pipelines, e.g., to add a file to a Dropbox folder or to send an email message. The integration of specific external functionalities, e.g., through manually coded APIs, is not included.

Table 11: Quickbase functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• no conceptual representation of functions available	• inaccessible	functions are specified according to Quickbase offered GUI screens
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• general, database-centric operations pre-implemented • no custom specification beyond system-specified options supported • no domain-specific reference functions available	• inaccessible	• implementation-level documents cannot be accessed • limited set of external functions can be accessed

Dynamic Perspective. In Quickbase, workflows that serve to integrate different data sources are called "pipelines". Pipelines consist of "steps", each of which incorporates a "channel" (see *Functional Perspective*). A step can be succeeded either by a further step or an if-else condition (see figure 8) – no further concepts for pipeline modeling are available. Surprisingly, no channels for database connections are included, only flat data files (e.g., CSV) can be integrated.

According to Quickbase's own documentation, these "pipelines" shall contribute to data integration.[13] It appears that pipelines mainly serve the purpose to provide some set of basic functionalities across Quickbase applications.

[13] https://university.quickbase.com/series/data-integration, accessed 08-04-2021

Table 12: Quickbase dynamic perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• generic process models	• proprietary	• diagram editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• no reference pipelines available	• "Steps" that contain channels • Conditions	• implementation-level documents cannot be accessed

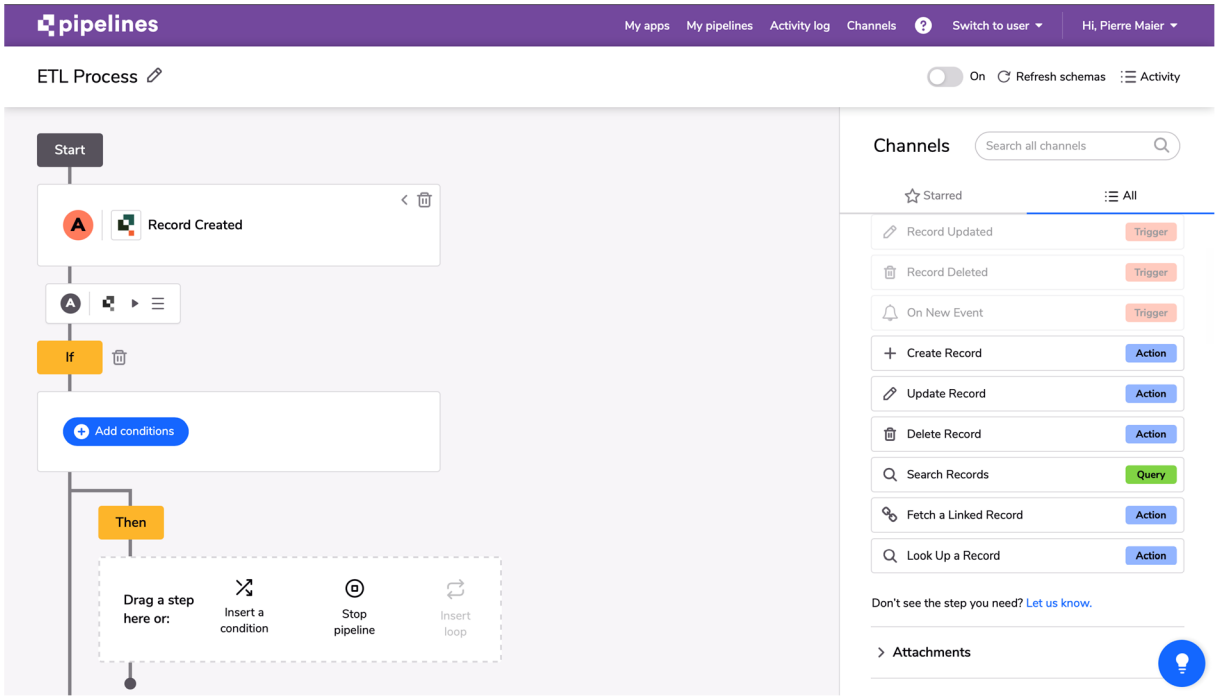

Figure 8: Quickbase pipeline example

GUI Development. As elaborated in the *Static Perspective* analysis category, every Quickbase application consists of at least n+2 pages for n user-specified entity types. The auto-generated pages for a single entity type display the persisted data of the respective entity type in tabular form. These GUI "pages" can contain subpages that either display user-defined charts of the data or provide a data entry form (see figure 9). A standard entry form is auto-generated based on the defined attributes of the entity type. This standard entry form can be adjusted by the user with regards to, e.g., grouping input fields in section, changing the order of input fields, or adding conditional rules for the visibility of input fields (e.g., to show input field A only when input field B is not null). The "Home" and "Users" GUI pages cannot be edited. It is also

possible to add an optional number of GUI pages, whereby these can also be specified via HTML and CSS.

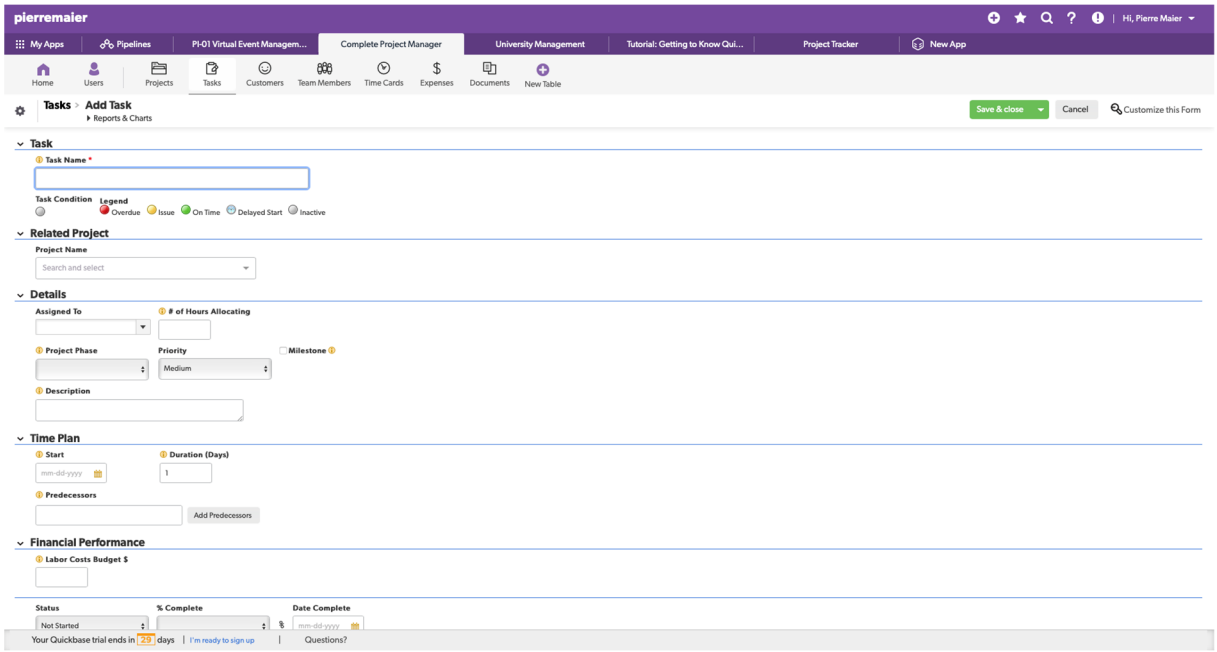

Figure 9: Quickbase exemplary add entry screen

Table 13: Quickbase GUI development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*
• GUI pages as a "tab" of an application	• GUI Editor • navigation pane cannot be adjusted	• each entity type is associated to one (system-defined) GUI page and vice versa • implementation of MVC pattern, with "controller" facet being largely inaccessible
Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• example applications can be adapted	• fields can be arranged and adjusted just in self-developed applications	

Further Aspects. Deployed Quickbase applications are only accessible within the platform itself. An organization is provided with a subdomain to access the acquired instance of the Quickbase platform. The web-based platform, as well as all applications developed with it, can therefore be accessed through the same URL. For mobile devices, Quickbase also offers a Quickbase app to access the developed applications. Quickbase applications cannot be exported as a whole, only the user-defined, persisted data can be exported through known formats such as CSV.

The Quickbase platform must be managed by one so-called "Quickbase administrator", who can invite users to and remove them from the platform. This Quickbase administrator also assigns new users to Quickbase applications in order for them to access and edit them. Users who create new applications are per default the respective application administrator. These assign the roles for users of the applications. Roles are specified according to the common CRUD scheme, which can be specified for each entity type separately. Additionally, different roles can be provided with different GUI pages, e.g., to show different charts to a project manager as opposed to a team member. Other than a definition of roles and users for application development, we could recognize no further collaboration support for developers and end-users.

Quickbase does not offer any AI services within the platform nor any integration of external AI services.

Table 14: Quickbase further aspects summary

<table>
<tr><td colspan="2">Accessibility and Convenience of Use</td></tr>
<tr><td colspan="2">• no specific methodical support for development
• fairly convenient and easy-to-use
  ○ GUI pages can be adjusted according to different user roles
• use of largely inaccessible, proprietary modeling languages</td></tr>
<tr><td colspan="2">Modes of Use</td></tr>
<tr><td>Platform (i.e., Application Development)</td><td>Application</td></tr>
<tr><td>• CRUD-based right specification (application users as application developers)
• no further support for collaborative application development</td><td>• CRUD-based right specification
• scarce integration capabilities between applications
• no further support for collaborative use of platform</td></tr>
<tr><td colspan="2">Focus on Deployment and Scalability</td></tr>
<tr><td>Target Location</td><td>Support</td></tr>
<tr><td>• cloud-based platform, accessible via a regular web browser
• data persisted on platform</td><td>no further accessible support mechanisms</td></tr>
<tr><td colspan="2">Focus on Artificial Intelligence</td></tr>
<tr><td colspan="2">• no AI services included</td></tr>
</table>

5.1.1.3 Quickbase: Conclusion

Emphasized Areas of Application Development. The identified features of the Quickbase platform mostly focus on static aspects of application development. This is exemplified through the provided visual data model editor and pre-defined business-oriented data types. Dynamic aspects are only partly accounted for through the use of so-called "pipelines", where system-developed connectors and event listeners can be accessed and executed. However, common

workflow modeling concepts are not considered: It is not possible to assign users to tasks, to monitor workflows at runtime, or to add connections other than the system-offered ones. Similarly, the platform lacks mechanisms that support the specification of functions. Collaborative development among different developers is also not addressed.

Provision of Abstractions. The focus of Quickbase platform lies more on abstracting away implementations details rather than to a to account for domain-specific use cases. Although some general, business-oriented data types are provided to specify entity types, they hardly provide productivity increase for more specific business domains. The small number of "templates" to define data models are also no reasonable candidates to contribute to more efficient application development. Additional aspects relevant for application development (e.g., data redundancy) are not accounted for with the obvious risk of producing inconsistent system states.

Role of IT Professionals. It is conceivable that most features of the Quickbase platform are generally accessible to lay developers, which conforms to their marketing focus on "citizen developers". Unreflective use of the platform features might, however, lead to inconsistent system states. Examples for this are the lack of data integration across Quickbase applications as well as the incapability to defined one-to-one associations in a data model. To manage the limitations of the Quickbase platform, more experienced developers would be required. The marketed image of Quickbase, which focuses exclusively on use by lay developers, is therefore misleading.

5.1.2 TrackVia

TrackVia is offered by a vendor of the same name. The company is based in Denver, CO.

5.1.2.1 TrackVia: Appearance of Vendor

Product Portfolio. The low-code platform of TrackVia is the only product offered by the company.

Product Provenance. The first archived web page of TrackVia is from Jul 16, 2006, where its platform is marketed as a "web-based database" for use by "any group of people". In particular, the platform is advertised as superior to spreadsheets solutions with regards to the scope of included functionality and "sophisticated software" solutions in terms of comprehensibility. In 2008, the TrackVia platform is described as a "form-maker to easily create web forms" (Jul 01, 2008). Later marketing statements refer to the platform as an online database with web forms and pre-defined templates (Mar 30, 2009). At a later point, it is also possible to identify slogans that indicate a clear correspondence to the current low-code trend: The TrackVia platform is referred to as an "easy-to-use application builder in the cloud" (Sep 02, 2011) with "0% coding required" (Mar 16, 2012) for "business people to create and use their own […] applications" (Mar 08, 2013). Without any apparent changes in the platform's features, it is relabeled as an "online workflow software" on Apr 02, 2015, and as a "low-code application software"

on Jun 03, 2017. Currently it is alternatively referred to as "no-code/low-code platform"[14] and "no-code app builder"[15].

Focus on Marketing. On the TrackVia web page, "low-code" is associated with faster development and increased ease-of-use.[16] The platform is marketed to provide support for "citizen developers" through "drag-and-drop configuration"[17]. It appears that the mainly promoted selling proposition for the TrackVia platform is to provide central data management as opposed to managing data through a stack of papers or numerous distributed spreadsheets. Advertised use cases are field services, project management, or also supply chain management, among others.

Table 15: TrackVia profile of vendor summary

Product Portfolio
• LCP only product of TrackVia
Product Provenance
• emerged as "web-based database" for "any group of people" • over time increased focus on "application development" • low-code label adopted in 2017
Focus on Marketing
• increased "ease-of-use" and development speed • centralized data management as advantage over spreadsheet solutions • focus on citizen developers • project management, field services, supply chain management as use cases

5.1.2.2 TrackVia: Analysis of Platform Features

The home screen of the TrackVia platform presents an overview of all available applications (see figure 10). The "Pluto's Norway" in the top left corner is an exemplary logo and company name that can be adjusted. Each TrackVia application consists of nine GUI pages: "Tables",

[14] https://trackvia.com/software-apps-low-code-development/, accessed 08-06-2021

[15] https://trackvia.com/, accessed 08-06-2021

[16] https://trackvia.com/software-apps-low-code-development/, accessed 08-06-2021

[17] https://trackvia.com/it-professionals-app-building-platform/, accessed 08-06-2021

"Views", "Forms", "Dashboards", "Filters", "Roles", "Table Relationships", "Flows", and "App Overview", the latter of which is displayed in figure 11.

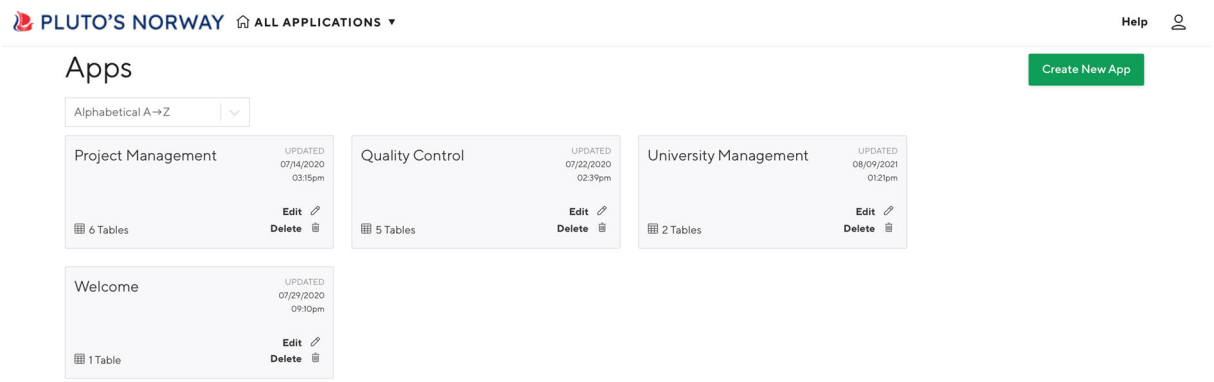

Figure 10: TrackVia home screen

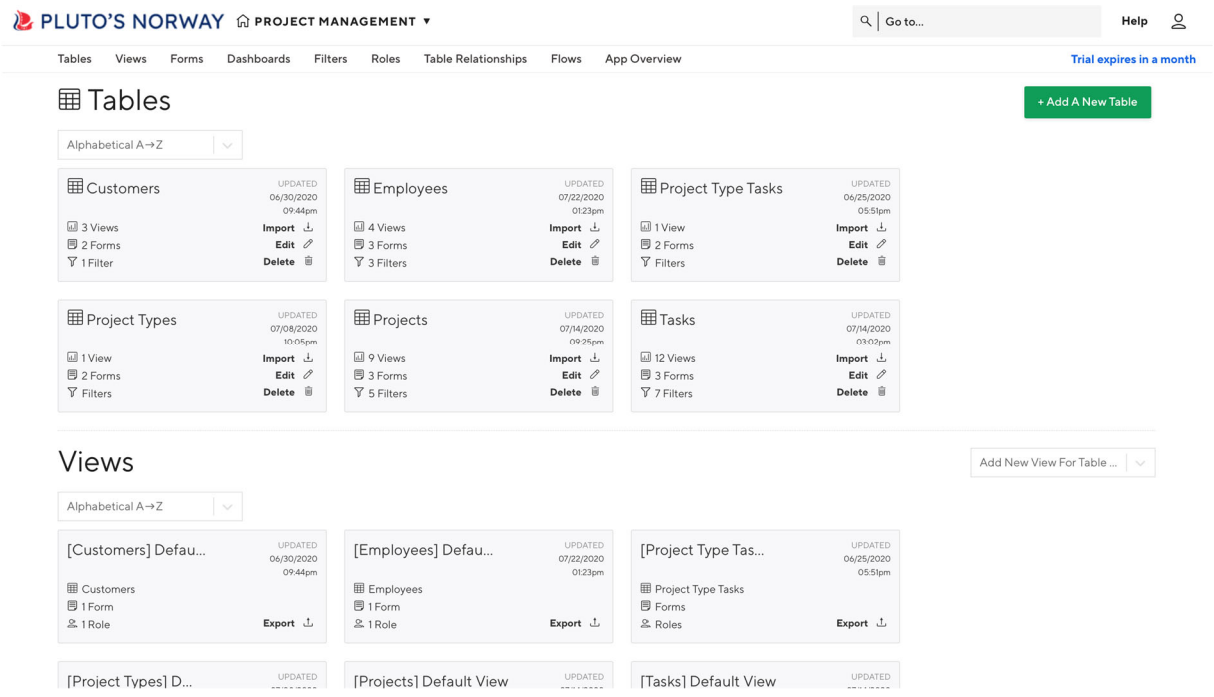

Figure 11: TrackVia application overview example

Static Perspective. To specify entity types, several system-defined data types can be used within the TrackVia platform. Some of them are generic text and number types, others are tailored

towards general business use like currency or location. It is not possible for a user to define additional data types. Entity types can be referenced across different TrackVia applications. Each TrackVia application is based on a single data model which can be viewed graphically in an ERD-like notation in the "Table Relationships" GUI page (see figure 12). Association types can be named. Role designators cannot be added. Only one-to-many associations can be defined, whereby multiple associations between two entity types can be added. According to TrackVia, users can add multiple one-to-many associations to replace many-to-many assocations in some cases, e.g., three one-to-many associations shall represent one three-to-many association. Such a workaround therefore proposes to fix the cardinality of a many-to-many association on one end. The persistence of data is handled by the TrackVia platform. The persistence mechanisms as well as the database schema are not accessible. Data reference models are not available.

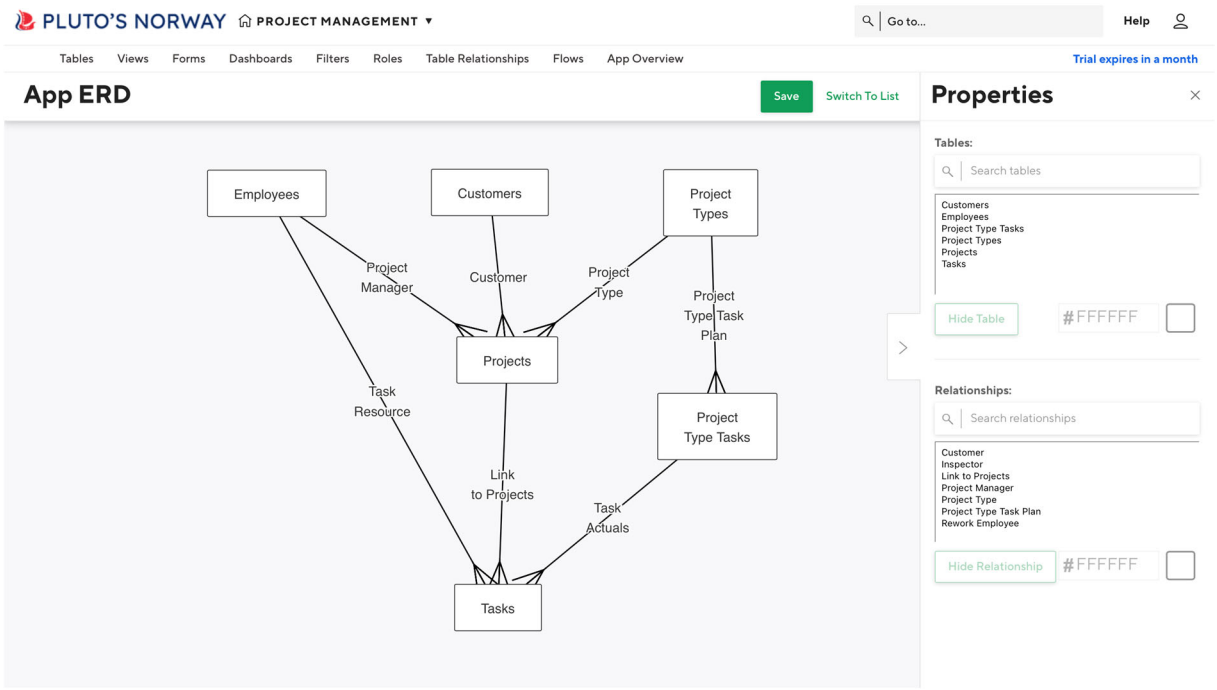

Figure 12: TrackVia data model example

Table 16: TrackVia static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• data model	• proprietary language similar to ERM	• diagram editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• general data types for numbers, strings, etc. • some more specific data types, e.g., for currency, users, or locations • no reference data models available	• (multiple) one-to-many relations between entity types	• no access to implementation-level documents
Focus on Integration		
Common static abstractions across a range of applications	single entity types can be referenced across applications	
Access to common data repositories across a range of applications	underlying data repositories inaccessible	

Functional Perspective. The TrackVia platform offers users two options to specify customized functions. The first option is to add so-called "formula" attributes to user-defined entity types. The values of "formula" attributes are calculated based on a set of predefined formulas, which resemble spreadsheet functions. The value of a regular "formula" attribute is calculated anew each time it is requested (e.g., for charts). Users can also define a formula attribute as "triggered", in which case the value of an attribute is only calculated when a respective entity is created, updated (i.e., either its type definition or some values of the entity), or deleted. The second option for functional specification is the implementation of so-called "Application Scripts". These user-defined scripts are written in the Groovy programming language. A source code editor can be accessed within the TrackVia platform (see figure 13). References to entity types in application scripts are possible but do not update automatically – if the attribute name of an entity type is altered, it has to be updated manually in every affected script. It is also worth noting that TrackVia's application scripts cannot be used to reference and execute other application scripts within the platform. The execution of each application script must be triggered by some system-defined event, whereby only database-related events (e.g., update entry, insert entry) are offered.

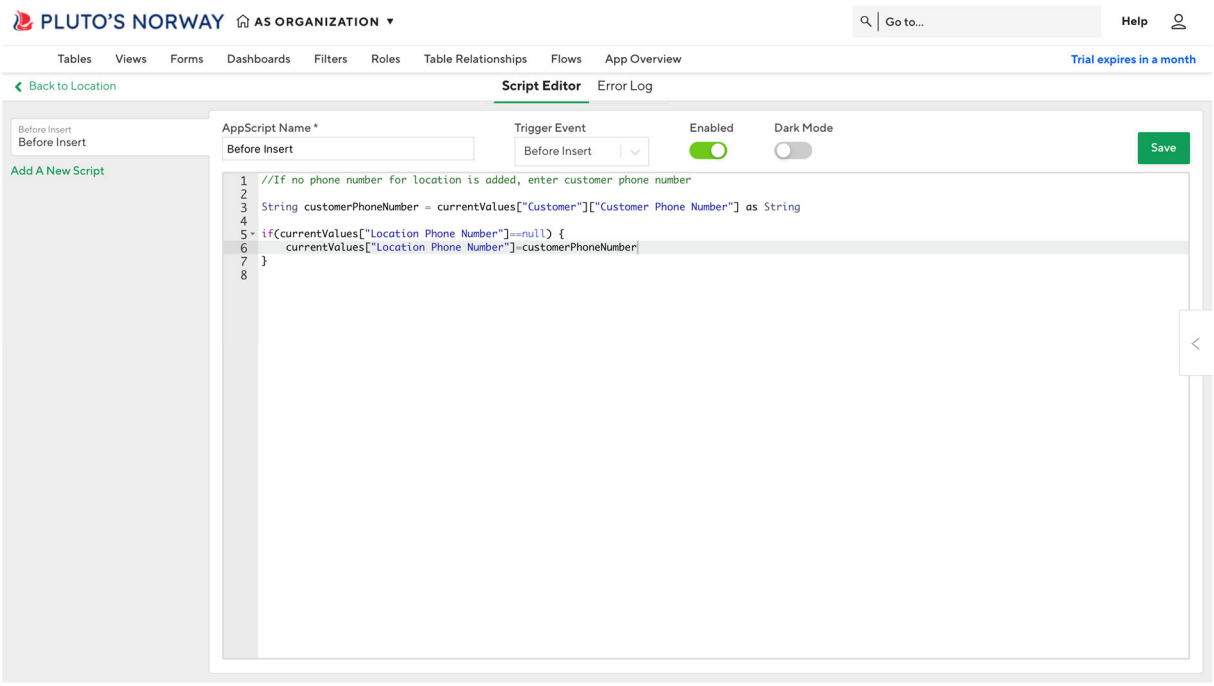

Figure 13: TrackVia app script example

Table 17: TrackVia functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*

• *no conceptual representation beyond code*
  ○ spreadsheet-like formulas
  ○ application scripts
	• proprietary language for formulas
	• Groovy programming language for scripts
		• code editor for app scripts

Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic arithmetic functions • no further reference functions available	• composition	• implementation-level documents beyond app scripts unavailable

Dynamic Perspective. Within the TrackVia platform, two distinct modules explicitly address dynamic aspects of application development. One is a separate integration platform called workato[18], the other is the "Flows" page available within each TrackVia application. Although workato is embedded within the TrackVia platform, since it is technically a separate platform

[18] https://www.workato.com/, accessed 08-10-2021

which does not offer any trial version, the functionality of workato for TrackVia applications cannot be adequately assessed. According to resources provided by TrackVia[19] connections can be based either on one of the already established connectors or through self-programmed HTTP, SQL, or SFTP requests.

The available "Flows" page gives access to editing und using "flows", which are also referred to as workflows by TrackVia. User-defined flows specify a sequence of GUI forms to view, add, and edit data entries. Flows are defined through a diagram editor, which distinguishes between "task", "decision", and "start/end" (see figure 14). "Task" nodes correspond to some GUI form, where user-defined entities can be viewed, added, or updated. "Decision" nodes are used to define conditional statements. An exemplary start screen of a flow is displayed in figure 15. It appears that TrackVia's flow models only serve to automate navigation between GUI forms within one application. Reference flow models are not available.

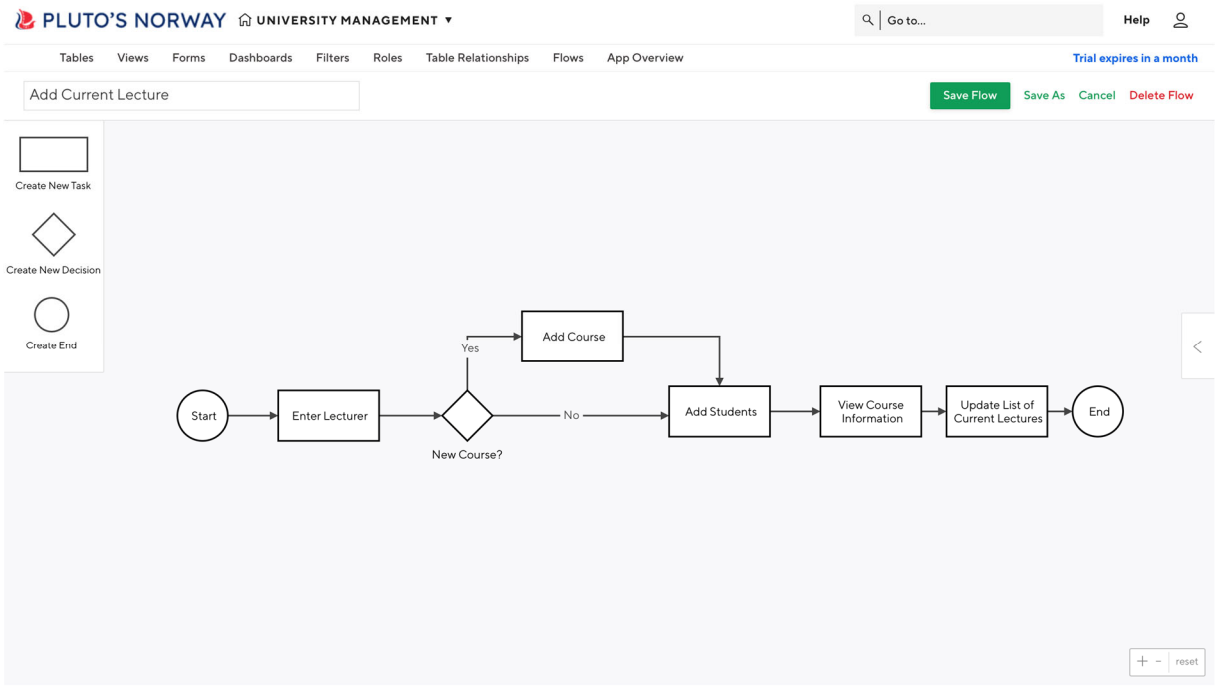

Figure 14: TrackVia flow example

[19] https://university.trackvia.com/integration-platform, accessed 08-10-2021

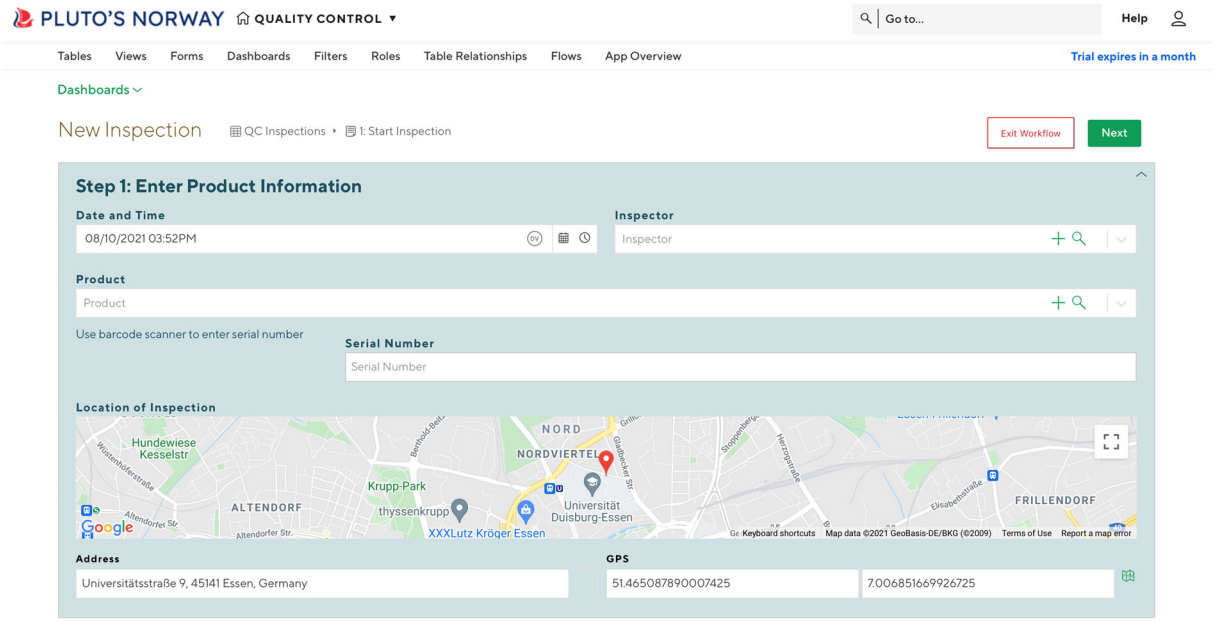

Figure 15: TrackVia exemplary flow start screen

Table 18: TrackVia dynamic perspective summary

Conceptual representations	*Languages*	*Components*
• Generic process model	• proprietary	• diagram editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• no reference abstractions available	• "Task" and "Decision" nodes	• no access to implementation-level documents

GUI Development. GUI pages can be designed in three tabs of the TrackVia platform: "Views", "Forms", and "Dashboards" (see figure 11). In "Views" and "Forms" users are provided with a drag-and-drop GUI editor. Charts can be added in "Forms" only. A dashboard is any combination of views and forms on a single GUI page. Reference GUIs are not available.

Table 19: TrackVia GUI development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*
• GUI models	• Drag-and-drop GUI editor	• Implementation of MVC pattern
Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• *none available*	-	

Further Aspects. The cloud-based TrackVia platform is accessible through a regular web browser. TrackVia also offers a separate mobile application of its platform for Android and iOS, which shall optimize interaction with developed applications on mobile devices. Applications as a whole, and thus also any specified application scripts and flows, cannot be exported. Single forms may be exported through a "Webform Generator" in HTML format[20], single tables as CSV files. Exported tables do not include any related data from other entities. External applications can interact with the TrackVia platform via a TrackVia API, that provides GET, POST, DELETE, and PUT operations.[21] Within TrackVia, the use of this API is referred to as a "microservice".[22]

One so-called "super administrator" of the platform assigns users to applications. Given the provision of access to a certain application, no further CRUD-based distinction of rights is possible. This also means that users of an application are not distinguished from application developers.

TrackVia does not include any internal or external AI services. We could not determine whether this is the case for workato, too. We can only speculate that AI services would likely be advertised with workato if they existed.

[20] https://developer.trackvia.com/downloads/webform-generator/, accessed 08-11-2021

[21] https://www.npmjs.com/package/trackvia-api, accessed 08-11-2021

[22] https://developer.trackvia.com/microservices/, accessed 08-11-2021

Table 20: TrackVia further aspects summary

Accessibility and Convenience of Use	
• no methodical development support • fairly convenient and easy-to-use • no adaption to particular user groups • use of largely inaccessible, proprietary modeling languages	
Modes of Use	
Platform (i.e., Application Development)	*Application*
• access to applications granted by "super administrator" (application users as application developers) • no direct access to external database schemata • no further support for collaborative application development	• no distinction of CRUD rights • no further support for collaborative use of platform
Focus on Deployment and Scalability	
Target Location	*Support*
• cloud-based platform, accessible via a regular web browser	• outside accessibility through RESTful API
Focus on Artificial Intelligence	
• no AI services included in TrackVia	

5.1.2.3 TrackVia: Conclusion

Emphasized Areas of Application Development. Although TrackVia offers separate features to handle static, functional, and dynamic aspects of application development, the platform's focus is clearly on static aspects and GUI design. The execution of user-defined functions is restricted to system-defined events. Workflow modeling features are, as far as accessible, rudimentary and limited to navigation of GUI pages. Further application development lifecycle activities like, e.g., testing or requirements engineering are not explicitly addressed.

Provision of Abstractions. Domain-specific abstractions are only scarcely embedded in TrackVia through the provision of some general business-oriented data types like currency. Productivity increase is realized through fading out implementation-specific peculiarities. Examples for this are the flow diagram editor or the drag-and-drop GUI editor. Users of the TrackVia platform cannot access or edit any underlying implementation documents.

Role of IT Professionals. On its web pages, TrackVia explicitly advertises to provide support for so-called "citizen developers". While most features considered in our analysis seem generally accessible to lay developers, the specification of application scripts can be noted as an exception here. The lack of cross-references in application scripts, which is likely to contribute to

functional redundancy, and the lack of dynamically updating embedded references, clearly restricts productivity increase through TrackVia's application scripts.

5.1.3 "Low-Code" Basic Data Management Platforms: Conclusion

One aspect seems especially interesting among the two examined LCPs: both vendors advertise an apparent superiority over spreadsheets and paper-based data management. The promises made by the vendors also appear constant over time: quicker application development, central data management to cope with dispersed information, and easy development even for "everyday business people" (Quickbase) and "any group of people" (TrackVia).

Our analysis of platform features proposes that both platforms emphasize a static perspective on applications. Features of the functional and dynamic perspectives are not completely disregarded, but only considered in a limited sense. The use of drag-and-drop editors for GUI pages and a form- and model-based development approach, partly embedded in the considered LCPs, is also hardly novel. Opposed to such earlier approaches, however, Quickbase and TrackVia are partly tailored towards generic business needs as showcased by, e.g., system-defined data types such as address or currency.

In both LCPs, users can scarcely access and edit source code, which limits the range of possible adjustments to the provided mechanisms of the platform. The limited export capabilities, i.e., mostly to spreadsheet files, also increase vendor dependency, which poses a clear risk to the protection of investment. Aside from these risks, limitations, and challenges, both platforms do allow for a quick development of simple data-centric applications without much experience in database design or application development. That is however not to suggest to mitigate the role of professional developers as proposed by the vendors' claims. The presented integration challenges and vendor dependency make the two LCPs no viable option for enterprise-wide applications.

5.2 Workflow Management Systems

LCPs in this category showcase a clear focus on workflow modeling capabilities and a historical development of the platform as centered around business process management.

5.2.1 Bonita

Since it was founded in France in 2009, Bonitasoft has had a clear focus on workflow management. This is reflected by the Bonita platform, which is still marketed as a process automation solution.

5.2.1.1 Bonita: Appearance of Vendor

Product Portfolio. The Bonita platform is the only product offered by Bonitasoft and it is available in a community and enterprise edition, the former of which is used for the subsequent analysis.

Product Provenance. The earliest archived home page of Bonitasoft is from Jun 25, 2009. Bonitasoft is marketed as "the open source BPM company" (Jul 26, 2009). In line with this proclaimed focus of the company, the Bonita platform is marketed to "generate, deploy and integrate process-based applications" (Sep 17, 2009)[23]. The notion of "process-based", or alternatively, "process-driven" applications is used continuously since then. The slogan "less code, more business value" is presented on Feb 29, 2012. Low-code is mentioned at least since Oct 13, 2017, as part of the platform labels "model-driven, low-code business process application development platform" and "low code application development platform". The latter is still in use today alongside the notion of a BPM platform.

Focus on Marketing. For Bonitasoft, the term "low-code" is used to denote application development platforms, which offer a set of reusable, out-of-the-box artefacts that can be adapted according to the developers' needs.[24] It is notable, that LCPs are thereby advertised as providing support for "development teams"/"the technical teams", and explicitly not lay developers (ibid.). Additionally, the Bonita platform is supposed to support "DevOps and Continuous Delivery"[25] as well as "Process Mining and AI"[26].

Table 21: Bonita profile of vendor summary

Product Portfolio
• LCP Bonita as only product of Bonitasoft
Product Provenance
• clear focus on BPM and "process-driven applications" since product emergence • denoted LCP since 2017
Focus on Marketing
• low-code marketed as support to tailor reference solutions for increase development speed and application quality • focus on "development teams", i.e., technologically-affine people

[23] http://www.bonitasoft.com/products/bonita-v4.php (archived)

[24] https://www.bonitasoft.com/low-code-application-development-platform, accessed 08-12-2021

[25] https://www.bonitasoft.com/bonita-continuous-delivery, accessed 08-12-2021

[26] https://www.bonitasoft.com/process-mining-AI, accessed 08-12-2021

5.2.1.2 Bonita: Analysis of Platform Features

The Bonita platform consist of three components: *Bonita Studio*, *Bonita Portal*, and *Form Designer*. *Bonita Studio* is the main development environment of the platform. It is a separate software system that supports the development and deployment of the so-called process-driven applications. Some elements of *Bonita Studio*, which are elucidated further below, can be connected to web-based GUIs. The development and design of web-based GUIs is handled in the *Form Designer*, the user-facing application can be accessed via *Bonita Portal*. The home screen of *Bonita Portal* is displayed in figure 16, where the registered user can access and execute open tasks.

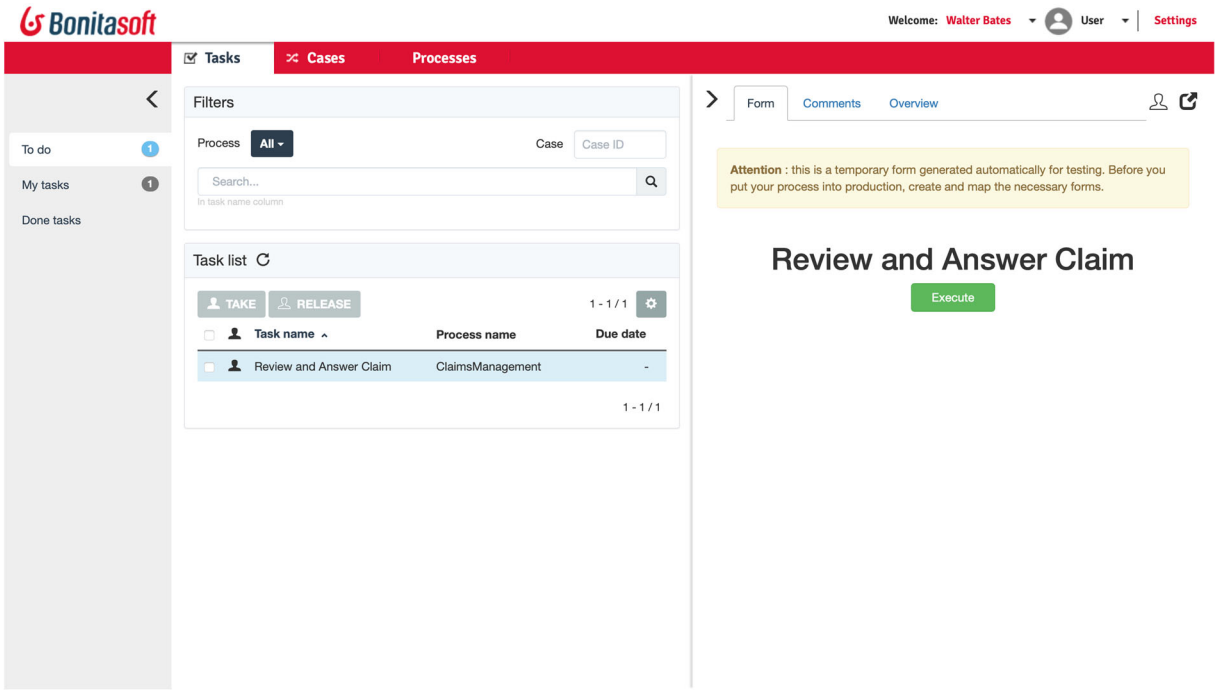

Figure 16: Bonita Portal

Static Perspective. The definition and maintenance of "business data models" is managed in *Bonita Studio*. Users of *Bonita Studio* can only interact with so-called "business objects", which is an overloaded concept that may represent entity types and classes, but also respective instance. These business objects are defined through a set of attributes, each of which is specified through a data type. Common, generic data types are offered for this purpose, among others, string, integer, Boolean, long, or float. The information provided in the business model editor GUI (see figure 17) suggest that business objects simultaneously serve to specify Java classes and entity types. For example, it is explicitly stated that the data type "TEXT" is mapped to a Java string and database character large object (CLOB). Generalization/specialization of these "business objects" is not possible. Once a business object is defined, it can also be used in the definition of other classes through an aggregation or composition relation. This provides developers an opportunity to associate business objects with one another. Reference models or any further domain-specific data types are not available on the platform. The data/object

model can also be edited via the corresponding XML file. The definition of entity types/classes can serve to instantiate objects for use in workflow models. The data generated in workflow instances (so-called “cases”) are persisted by Bonita itself on an H2 relational database. Data models cannot be viewed graphically, but database schemata are accessible via regular SQL queries (see figure 18).

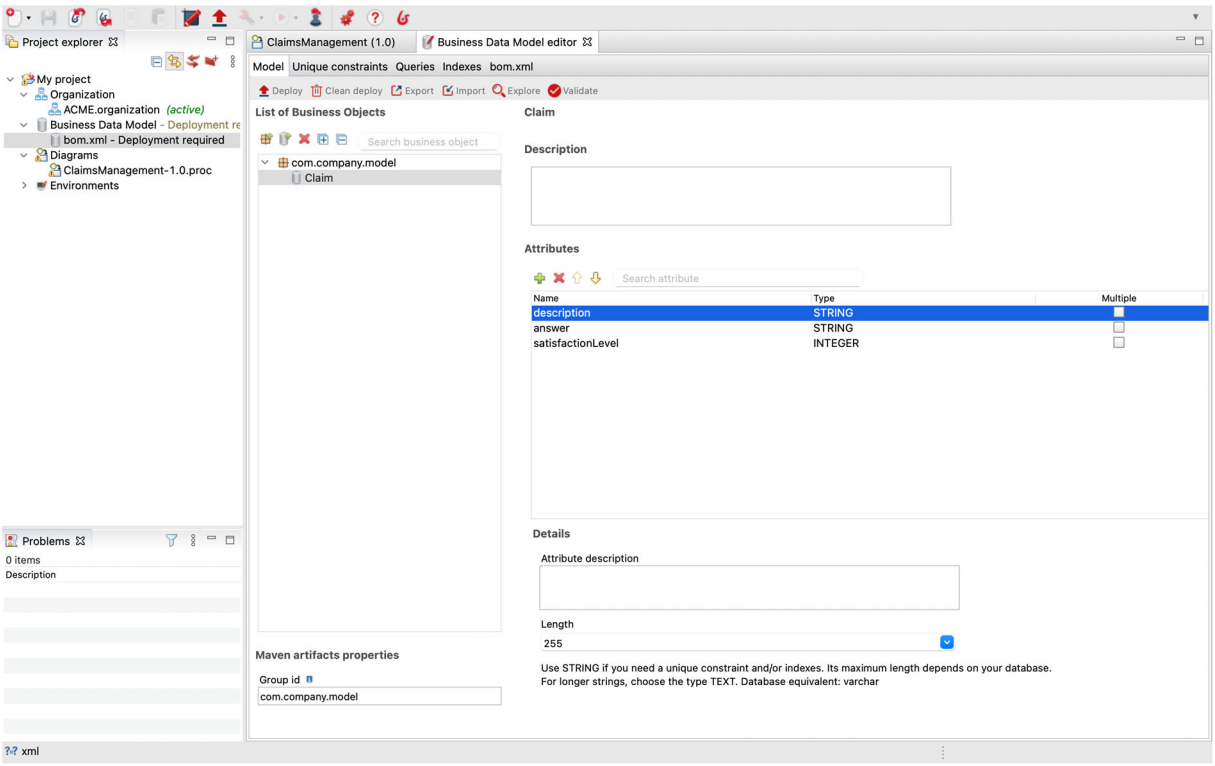

Figure 17: Bonita Studio business data model

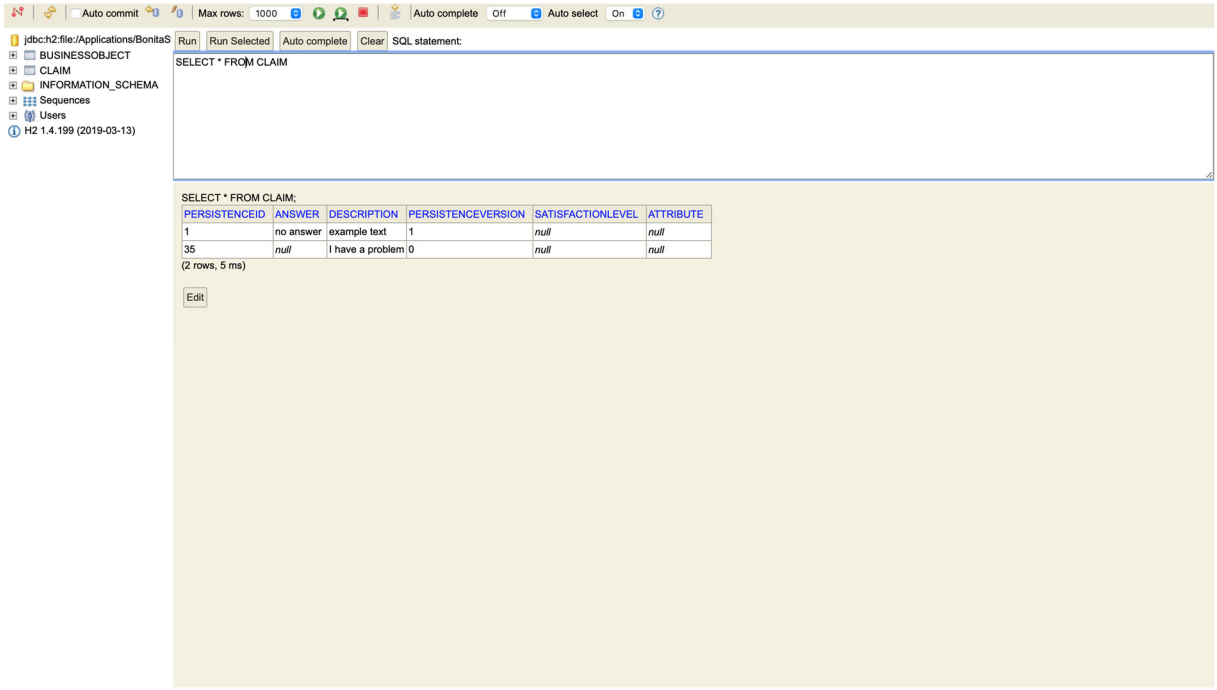

Figure 18: Bonita Studio SQL query on relational H2 database

Table 22: Bonita static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• data model	• proprietary, based on Java and SQL representations	• form-based GUI editor synced with XML file • generates Java classes and SQL statements
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• common data types like string, integer, float, etc. • No further reference data types or data models available	• aggregation/composition associations	• access to corresponding XML file • database can be accessed, possibility to define SQL queries
Focus on Integration		
Common static abstractions across a range of applications	use of common data models and integration thereof depends on specified workflow types	
Access to common data repositories across a range of applications	all data is persisted in one H2 relational database	

Functional Perspective. Bonita Studio does not offer any functional modeling component. Users can implement custom function through a built-in Groovy source code editor. The written scripts can be embedded in workflow models. Some basic functions to access nodes in a workflow or reference particular users are pre-implemented and accessible through the workflow engine API.

Table 23: Bonita functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• no representation beyond source code	• Groovy programming language	• Groovy editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• small set of pre-defined system functions for workflows	• *none*	• Groovy source code

Dynamic Perspective. Workflows in *Bonita Studio* are modeled according to the business process model and notation (BPMN) modeling language. An exemplary workflow for claims management is displayed in figure 19. An instance of a workflow model is referred to as a "case" within the platform. Each workflow type can include many "business variables", each of which is reference to some user-define "business object" (see *Static Perspective*). Business variables are embedded in workflows through "contracts" which specify the expected input data type at each node in the workflow. Application user roles (as "actors", see *Further Aspects*) can be assigned to task types and lanes. Furthermore, task types include a reference to a web-based GUI "form" (see *GUI Development*). Connectors to external systems can be defined at the input or output of a task type (see figure 20). These can serve to integrate the workflow with further systems according to common interfaces (e.g., through SOAP) or through self-programmed connectors.[27]

Table 24: Bonita dynamic perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• business process model	• BPMN	• BPMN diagram editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic process elements as included in BPMN notation • no further abstractions provided	• concepts from BPMN • business objects and contracts	• no access to implementation-level documents within Bonita Studio

[27] https://documentation.bonitasoft.com/bonita/2021.1/connector-archetype, accessed 08-16-2021

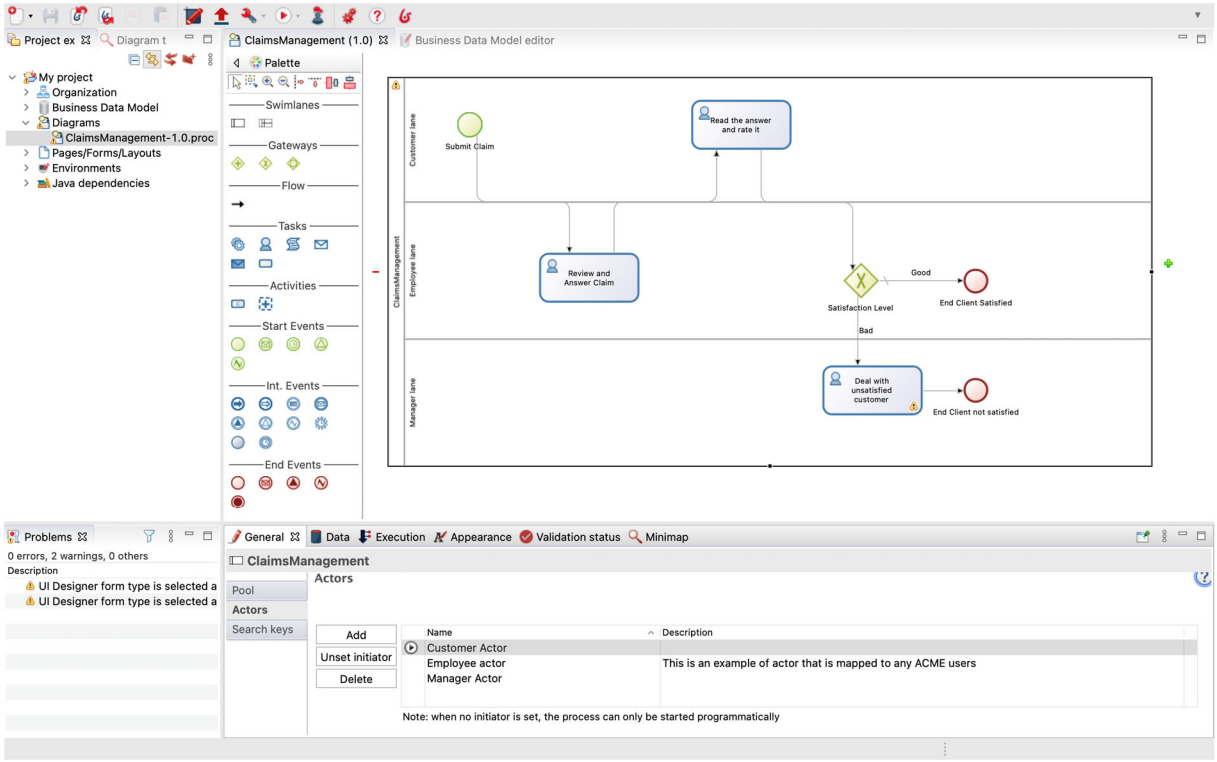

Figure 19: Bonita Studio example workflow model

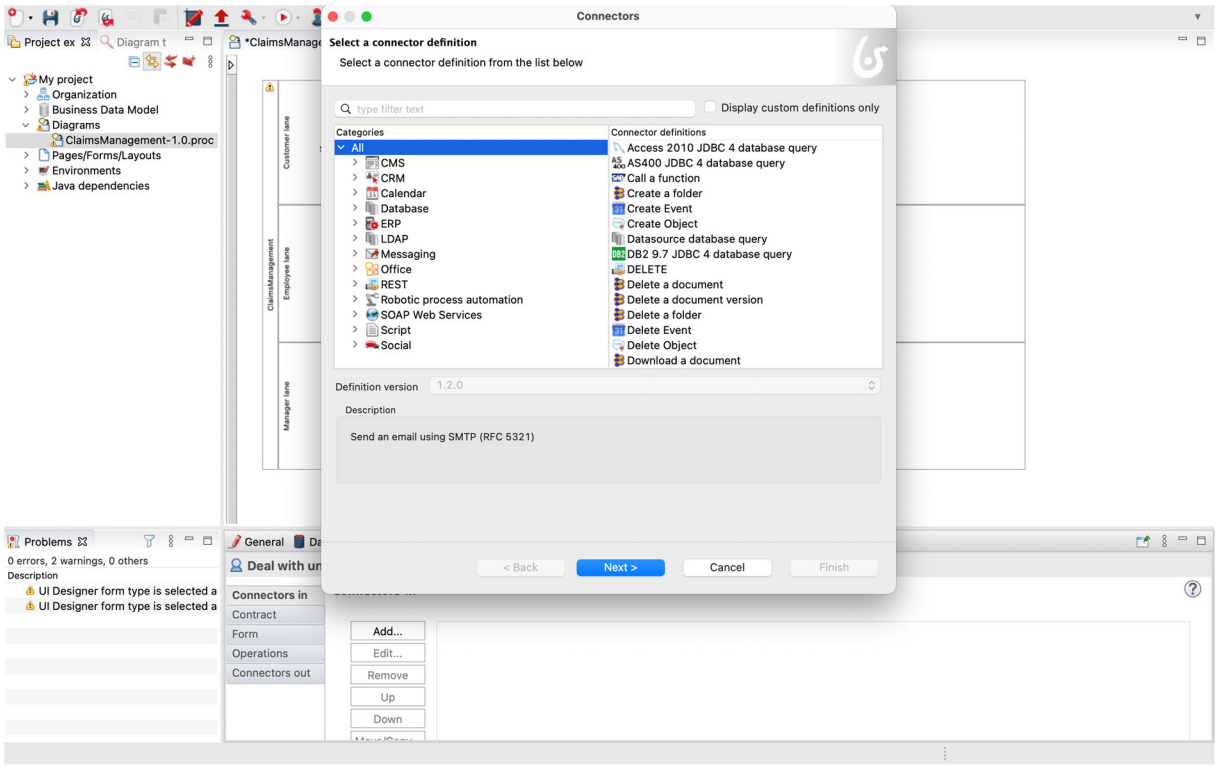

Figure 20: Bonita Studio workflow connector definition

GUI Development. Applications developed with *Bonita Studio* can be accessed through *Bonita Portal* – an example is displayed in figure 16. Applications focus exclusively on the execution and management of workflows specified in *Bonita Studio*. Therefore, only GUI pages which are referenced in the design of workflows can be viewed. GUI pages are generated automatically according to the defined contracts of a workflow type. Customization of GUI forms is enabled in the separate *Form Designer* module of the Bonita platform (see figure 21). Entire GUI pages beyond these "form" containers cannot be adjusted and must follow the system-specified style. Apart from GUI forms that serve to execute workflow instances, an "application page" can be specified which lists all past workflow instances. Reference GUIs are not included.

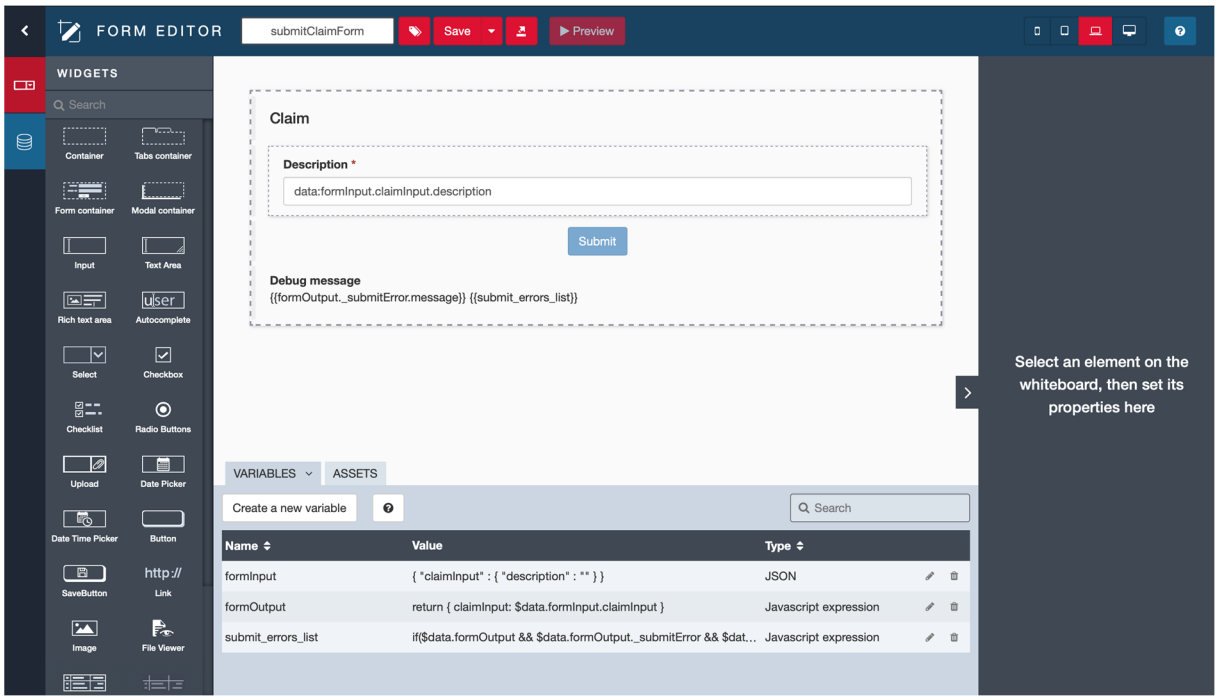

Figure 21: Bonita Form Designer

Table 25: Bonita GUI development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*
• GUI models consisting of "forms"	• "Form" Editor ○ drag-and-drop GUI editor	implementation of MVC pattern, strict correspondence to workflow model specified in Bonita Studio
Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• library of general GUI elements • example applications not available	• theme and parameters of GUI elements can be adjusted	

Further Aspects. Hosting of the relational database and workflow engine is managed by Bonitasoft. The server to host applications needs to be specified by the developers. All JAR, XML, and further files generated by *Bonita Studio* are freely accessible.

To define user roles, *Bonita Studio* allows to specify hierarchically arranged roles which shall loosely correspond to an organizational structure. This shall allow simple dynamic references to roles, e.g., "manager of X". Each application user is assigned to a certain role, but roles do not include any set of rights. The modification and access rights of roles is specified separately in each workflow model. *Bonita Studio* does not offer any support to specify roles and rights for developers of an application.

While Bonitasoft seemingly advertises AI services explicitly on their website as part of a "process mining" module[28], no such capabilities could be identified. The entire documentation of Bonita[29] does not yield any results for "process mining", "artificial intelligence", or "machine learning". It seems that neither internal nor external AI capabilities are currently included in the community edition.

Table 26: Bonita further aspects summary

Accessibility and Convenience of Use	
• no methodical development support • implementation and use split into three environments • implementation in Bonita Studio demands some familiarity with programming concepts • use of BPMN for workflows	
Modes of Use	
Platform (i.e., Application Development)	*Application*
• need for two supplementing environments: Form Designer and Bonita Studio • no specific support for collaborative development	• access to developed applications is granted through Bonita Portal • definition of hierarchical role structures with set of predefined interrelations (e.g., "is manager of") • roles are defined to a set of workflow elements within one lane
Focus on Deployment and Scalability	
Target Location	*Support*
• application hosting needs to be managed by developer	*no further accessible support mechanisms*
Focus on Artificial Intelligence	
• no AI services identified for Bonitasoft	

[28] https://www.bonitasoft.com/process-mining-AI, accessed 08-16-2021

[29] https://documentation.bonitasoft.com/bonita/2021.1/, accessed 08-16-2021

5.2.1.3 Bonita: Conclusion

Emphasized Areas of Application Development. The Bonita platform primarily offers support to model sophisticated workflows. This includes mechanisms to design and integrate GUI forms as well as to assign roles to workflow lanes. Capabilities to specify functions and data models are limited: custom functions can only be implemented via a built-in Groovy source code editor and visual representations of data models are not available.

Provision of Abstractions. Both *Bonita Studio* (generally for any static, functional, or dynamic specification) and *Form Designer* (only GUI design and integration with underlying functionality) do not provide domain-specific abstractions. No reference models are available. *Bonita Studio* allows a rudimentary representation of organizational structures through user and role definitions. However, this approach is not satisfactory in cases where an elaborate representation of organizational structures is required. Generally, the platform provides higher-level representations of a source code-based implementation (see, e.g., business object specification). This makes *Bonita Studio* hardly accessible to lay developers. Noticeably, abstraction from programming concepts is not only provided through visual modeling and input forms, but also through a simplified representation of standard programming concepts. The synthesis of classes and entity types as "business objects," and fading out of the respective object-relational mapping, might increase comprehensibility and productivity while restricting modification options for more experienced developers.

Role of IT Professionals. Bonitasoft explicitly targets professional application developers. With respect to *Bonita Studio*, it seems indeed reasonable to assume that only experienced developers will benefit, since familiarity with client-server architecture, object-oriented programming, and development with IDEs is required. Source code can partly be accessed and a built-in source code editor allows to define custom scripts. The *Form Designer,* which allows to develop GUIs per drag-and-drop, can also be used by lay developers since no advanced programming concepts are included here. However, no features to support collaborative development with lay developers can be detected.

5.2.2 Creatio Studio

Creatio Studio is offered by Creatio, a company that has offices in six countries. It is not clear, though, where its headquarters is located.

5.2.2.1 Creatio Studio: Appearance of Vendor

Product Portfolio. Creatio's product portfolio is somewhat opaque at first sight. On Creatio's home page, a distinct BPM and customer relationship management (CRM) component of the

same "Creatio platform" is advertised.[30] The former includes three products (*Creatio Sales, Creatio Service, Creatio Marketing*), the latter only one (*Creatio Studio*). The entire "Creatio platform" and *Creatio Studio* is marketed as an LCP. However, a "Creatio platform" is not available as such and only the four products listed above can be acquired. This begs the question what the denoted "Creatio platform" shall refer to and whether the three CRM products are also considered low-code by Creatio.

A closer look at each of the products clarifies this circumstance. Each Creatio product can serve as a stand-alone platform. Every platform includes two "workspaces" per default: one for application use (application workspace) and one for application development (development workspace). Creatio's CRM product line offers pre-implemented application workspaces tailored towards particular business needs. The development workspace is the same for all four products. It is thus also possible to acquire any combination of Creatio products and access them via the same Creatio platform – one would have then access to multiple application workspaces. The BPM *Creatio Studio* platform includes only the development workspace and aa generic application workspace (denoted as "applications") which does not offer any pre-implemented business application. At the same time, the development workspace is also called *Studio* in each of Creatio's products. Since the focus of our investigation is on LCPs, we will refrain from a closer look at particular CRM products.

Product Provenance. The company was renamed to Creatio only in 2019. Previously, it was called *bpm'online*.[31] The earliest archived web page of bpmonline.com from May 15, 2011 states that "BPMonline CRM is a comprehensive CRM solution which provides enhanced business process management (BPM) features". It seems that only one CRM product was offered at this time. An early product advertisement of its "Process Management Platform" (May 17, 2011)[32] describes a web-based platform that allows for integrating external sources and can be customized to specific requirements. On Aug 31, 2013 this platform is marketed as a solution for "process driven CRM" that shall enable "users to build applications they need at the click of a mouse" (Jul 01, 2015). "Out-of-the-box processes" and multi-channel application access is advertised on Feb 01, 2017. The dissemination into four different products (see *Product Portfolio*) can be noted at least since Aug 05, 2013. The entire catalog of products is still referred to as a single "platform", even though products can be deployed independently. Since at least Jun 30, 2019, the term low-code is used to market the platform. On this date, the platform itself is referred to as an "Agile Platform for Business Process Automation and CRM" with "ready-to-

[30] https://www.creatio.com/, accessed 08-18-2021

[31] https://www.creatio.com/company/news/17893, accessed 08-18-2021

[32] http://www.bpmonline.com/products/platform (archived)

use apps and templates". The renaming of the company from bpm'online to Creatio on Oct 30, 2019 is apparently accompanied by designating its platform as a low-code BPM platform.

Focus on Marketing. On Creatio's web pages, low-code is marketed as an innovation that serves to reduce development and improve communication between business and IT professionals.[33] The Creatio LCP is said to improve the adaptability of applications to a changing environment and is more intuitive to use than other platforms.[34] It shall enable everyone within an organization to develop the applications they need.[35] This indicates a marketing focus on lay developers.

Table 27: Creatio Studio profile of vendor summary

Product Portfolio
• Creatio Studio as LCP and business process management platform
Product Provenance
• previously named BPMonline • early focus on business process and customer relationship management • renaming of company to Creatio was accompanied by relabeling as LCP
Focus on Marketing
• quicker development time and improved business-IT alignment • focus on citizen developers

5.2.2.2 Creatio Studio: Analysis of Platform Features

The home screen of the *Studio* workspace is displayed in figure 23, the home screen of the *Applications* workspace can be seen in figure 24. Both constitute the *Creatio Studio*. For all workspaces, three GUI "panels" can be distinguished. The side panel on the left serves to navigate the workspace, the communication panel on the right serves to interact with other users via, amongst others, chat, phone, or mail, and the "main panel" (not designated as such by Creatio) comprises a search bar in the top and a working area underneath. This might be supplemented

[33] https://www.creatio.com/blog/do-you-really-need-learn-code-why-anyone-can-be-developer-low-code-platforms, accessed 08-18-2021

[34] https://www.creatio.com/blog/how-businesses-can-maximise-benefits-creatio-low-code-platform, accessed 08-18-2021

[35] https://www.creatio.com/our-technologies/low-code, accessed 08-18-2021

by a folder and filter area next to it. Both, the side and the communication panel, can be expanded or condensed by the user.

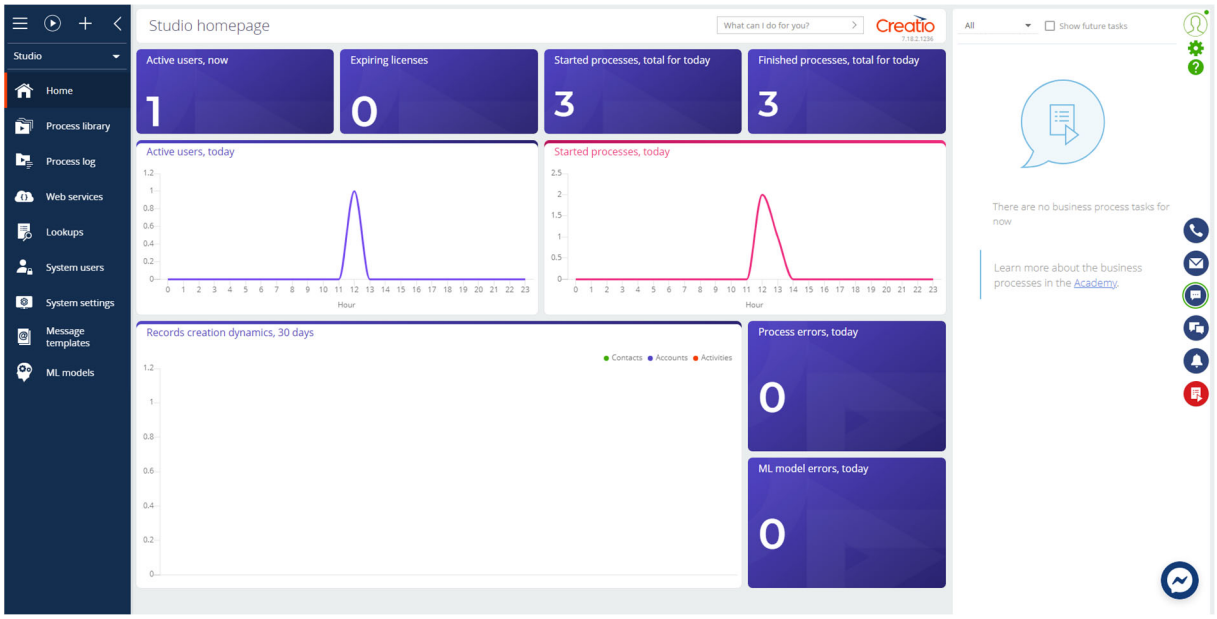

Figure 22: Creatio Studio "Studio" workspace home screen

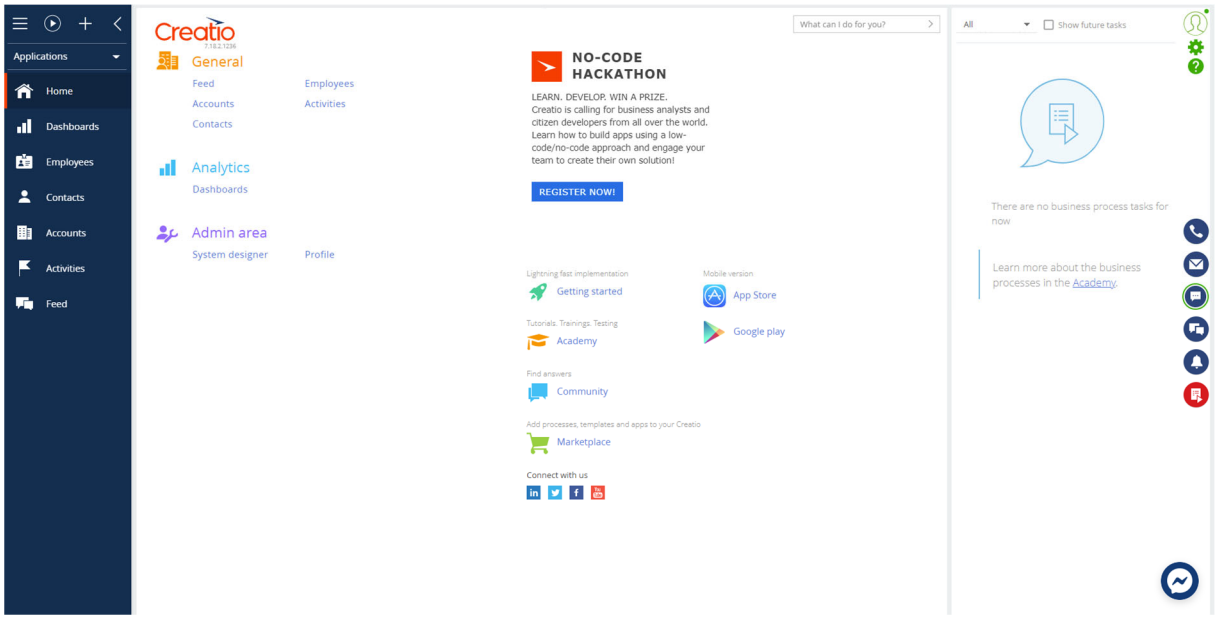

Figure 23: Creatio Studio "Applications" workspace home screen

Static Perspective. Three business-generic entity types are pre-implemented in *Creatio Studio*: employees, contacts, accounts, and activities. Per default, each of these constitute a section of the *Applications* workspace (see figure 23). There are two options to specify custom entity types. First, users can design GUI pages – the sections of the *Applications* workspace – per drag-and-drop that correspond to entity types (the "section page wizard" is displayed in figure 25). Each field of the GUI page corresponds to an attribute of the entity type. Therefore, the fields

must incorporate some specified data type. This includes generic types like Boolean, Integer, or String, but it is also possible to define "lookup" fields, which are a reference to other entity types within the *Creatio Studio* platform. Second, users can define entity types under the "System Administrator" area of the *Studio* workspace (see figure 24). Here, it is additionally possible to define "object inheritance" relations. The exact inheritance mechanisms are inaccessible. For both options, it is also possible to access and edit SQL and C# source code – which shows that the specification of entity types simultaneously serves to define C# classes. Visual data models are not available in *Creatio Studio*. Data persistency is realized through a relational database managed by *Creatio Studio*. Additionally, it is possible to search for and eliminate redundant data. The search for duplicates is dependent on user-defined rules, such as to remove person entries when the phone number and first name is identical.

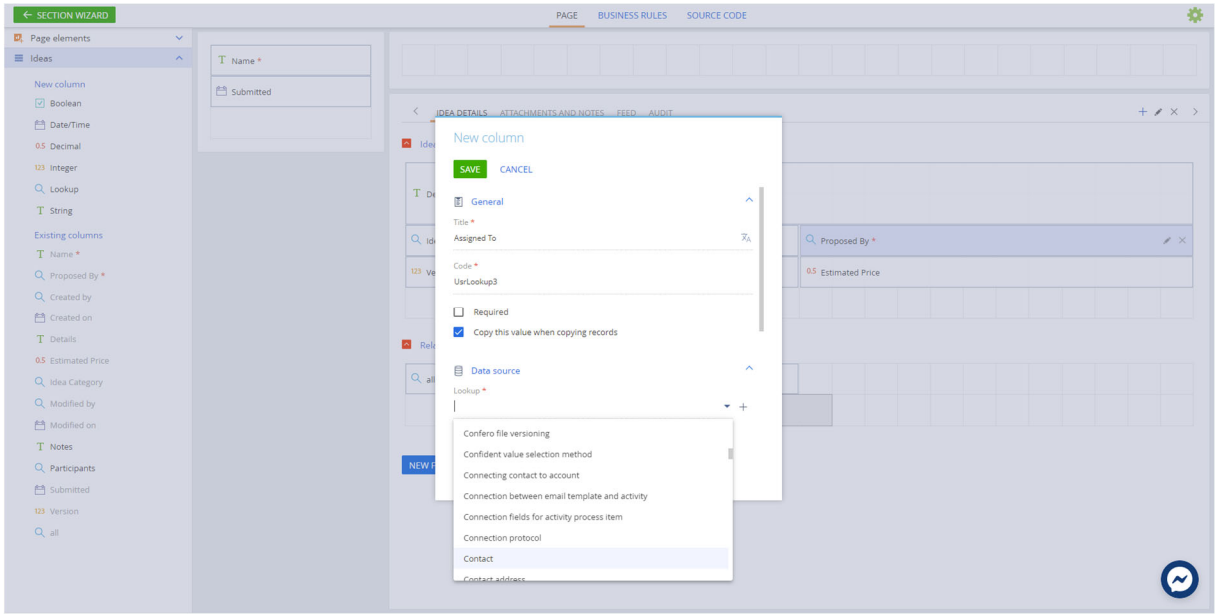

Figure 24: Creatio Studio section page wizard

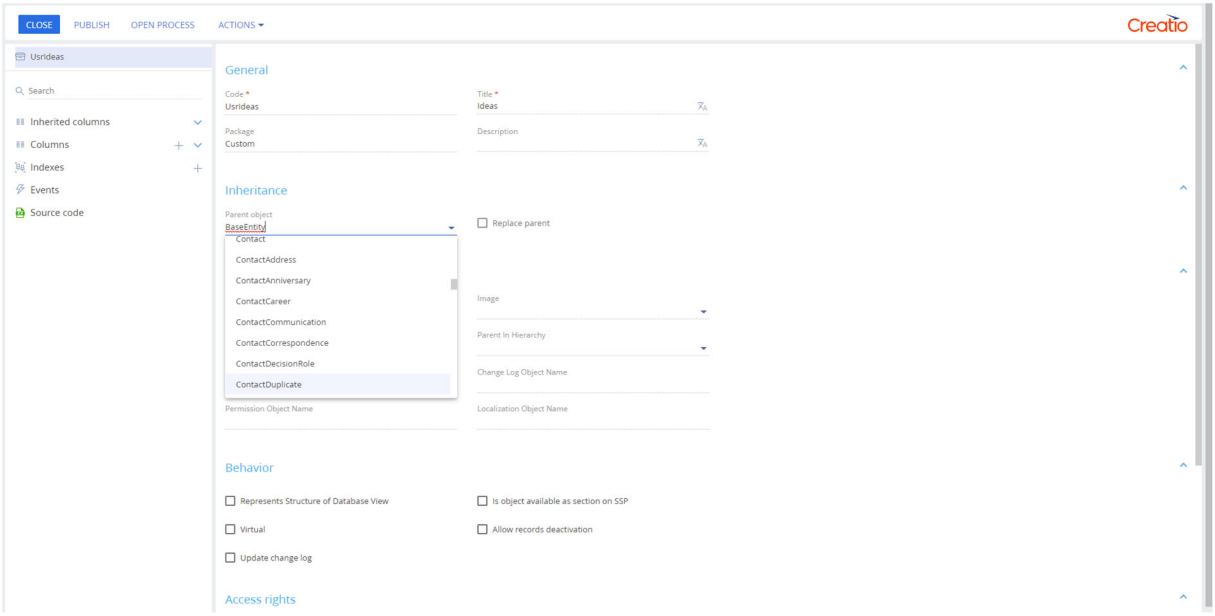

Figure 25: Creatio Studio "object inheritance"

Table 28: Creatio Studio static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• no conceptual representation beyond GUI designer and relational database system	• no visual modeling language provided • Classes are implemented in C#; database requests are in SQL	• user-friendly GUI for entity type specification • access to database system
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• common data types like string, int, etc. are provided • very restricted set of general business entity types can be adopted	• generalization/ specialization supported • references between entity types through "lookups"	• generated SQL code can be accessed and modified • corresponding C# classes can be viewed, but not edited
Focus on Integration		
Common static abstractions across a range of applications	Entity types provided by Creatio are necessarily part of every application. Beyond this, it is up to the application developer to make use of common static abstractions.	
Access to common data repositories across a range of applications	All applications access the same underlying database instance persisted on Creatio.	

Functional Perspective. Functions in *Creatio Studio* can be written in C# through a built-in source code editor. For this purpose, the LCP offers several libraries for implementation support. Among others, it supports the definition of customized event listeners for custom entity types.[36] The script-based, pre-implemented workflows (see *Dynamic Perspective*) can serve as reference functions. No dedicated component for modeling functions is available. Creatio also offers a marketplace where anyone can provide enhancements to the Creatio platform either for free or for some price (see figure 26). These enhancements are categorized either as a template, an add-on, a connector, or a "software solution". This is reminiscent of the "on-the-fly-computing" paradigm presented in chapter 2. Notable here are especially the various add-ons that are available. Add-ons comprise, among others, the "ATF.Repository for Creatio" for data modeling and explicit object-relational mappings, an add-on for creating pivot tables as Excel exports, or a command line interface for Creatio. "Software solutions" are complete Creatio applications (i.e., an application use workspace, see remarks in *Product Portfolio*) that include specific entity types, data schemas, domain-specific functionalities, and workflow models. Exemplary application domains are, among others, compliance or risk management.

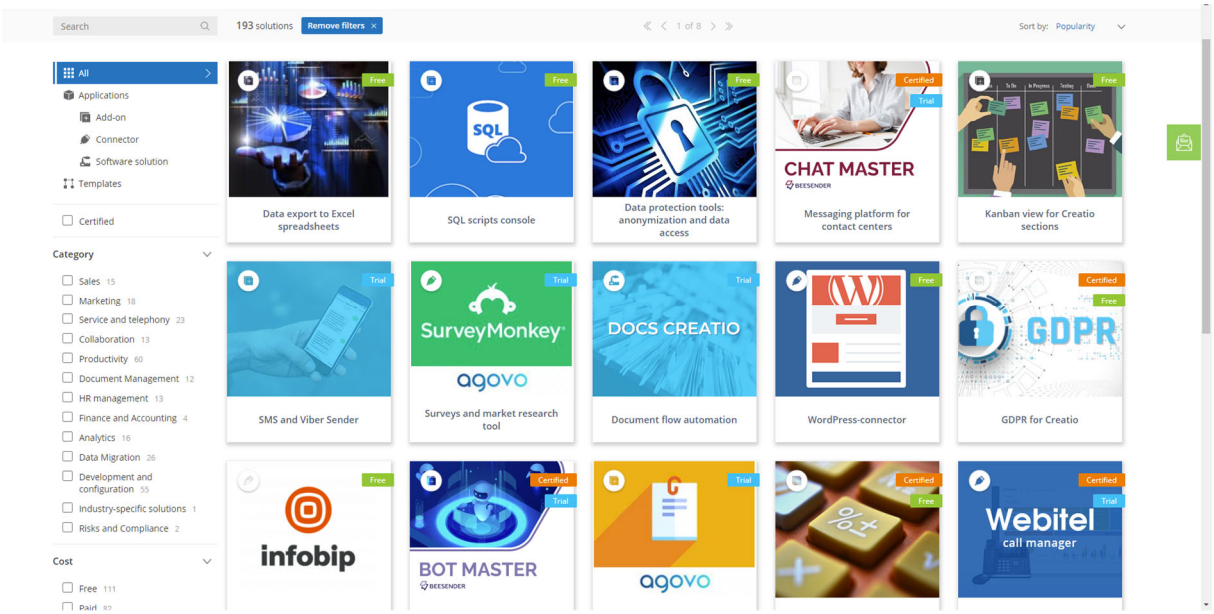

Figure 26: Creatio marketplace

Table 29: Creatio Studio functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*

[36] A complete list of available C# libraries can be explored at https://academy.creatio.com/api/netcoreapi/7.17.0/index.html, accessed 08-24-2021.

• *no conceptual representation for visual modeling of functions* • C#-based source code	• *none for visual modeling* • C#	• access to available C# source code
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• function libraries for general platform functionalities	• *no platform-specific language concepts*	• function libraries and source code can be accessed

Dynamic Perspective. According to Creatio, workflow types ("processes" in *Creatio Studio*) are modeled according to the BPMN 2.0 specification.[37] It is possible to define sub-workflow models which can be embedded in super-workflow models. A sizable number of workflows, most of which are based on custom scripts, are pre-implemented on the platform. These depict generic functionalities for using the platform, such as, e.g., "actualize contact age" or "bulk duplicate search". All custom or system-defined workflow models can be viewed and edited in the process library section of the *Studio* workplace (see figure 28). It is worth noting that Creatio's CRM products share the same set of system-defined workflow types as the *Creatio Studio* platform. For every modeled workflow type, the auto-generated C# code can be viewed (see figure 28). An example workflow diagram is displayed in figure 29. Entities persisted on Creatio can be accessed directly from the workflow designer. Remote API calls are enabled through the "call web service" task type which allows to define REST and SOAP-based requests via a pre-configured UI. C# code snippets can be embedded through a "script task" type and the "predict data" task type uses user-define ML models to make predictions (see *Further Aspects*). Domain-specific reference workflow models, as well as additional pre-configured connectors, can be downloaded via the Creatio marketplace (see *Functional Perspective*). In addition to workflows, so-called "cases" can be added to each section of the *Applications* workspace. Cases can assign a status to an entity. An example from the *Sales* workspace is displayed in figure 30, where a "lead" entity is currently in the "Handoff to sales" status (see status bar at the top).

[37] https://www.creatio.com/page/bpmn, accessed 12-19-2021

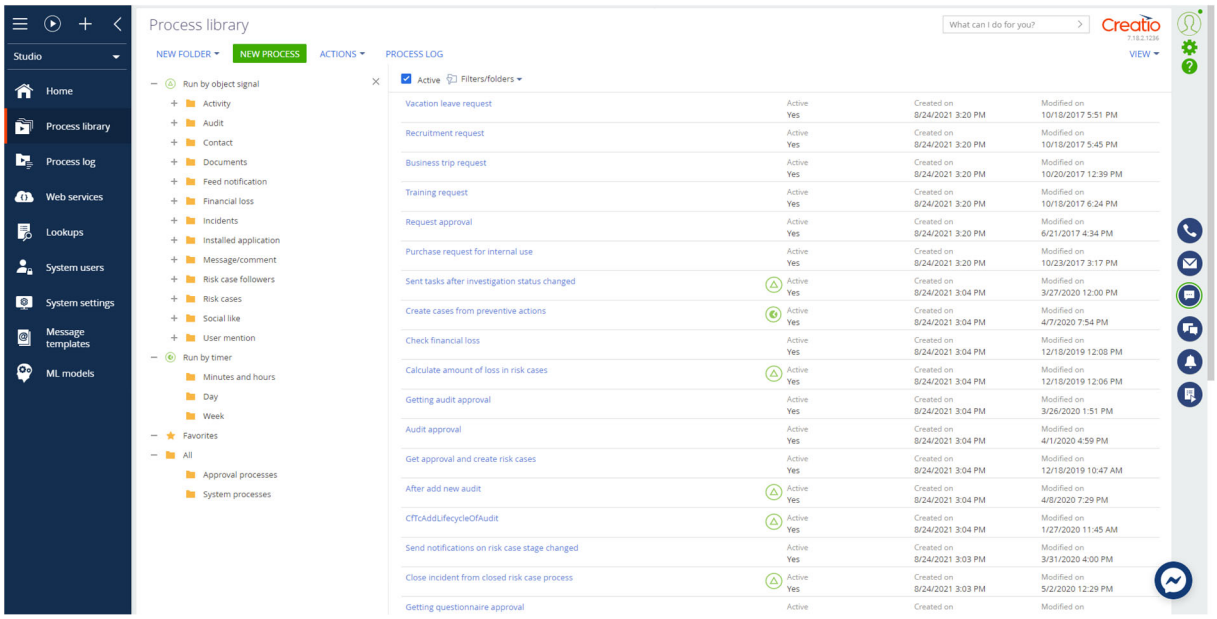

Figure 27: Creatio Studio process library

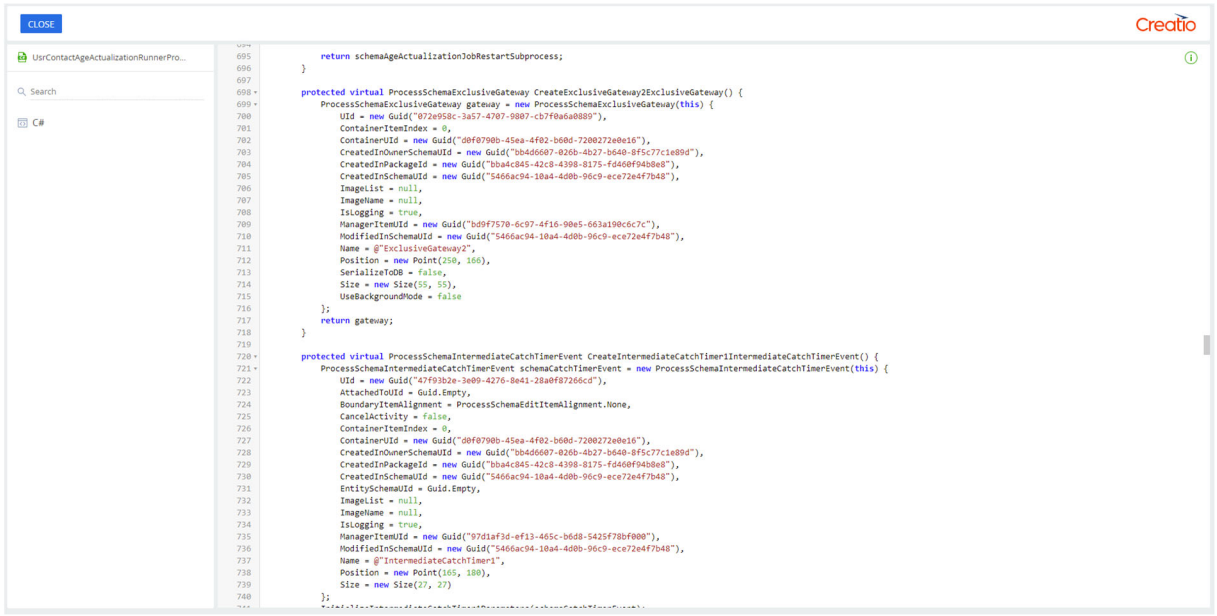

Figure 28: Creatio Studio workflow source code

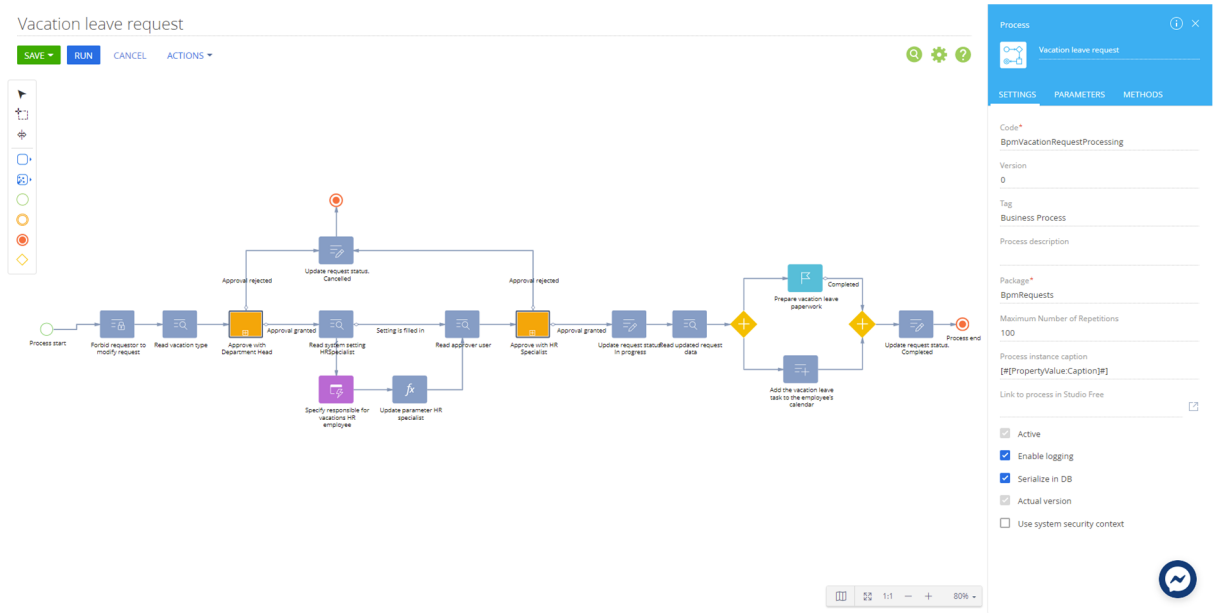

Figure 29: Creatio Studio workflow type diagram

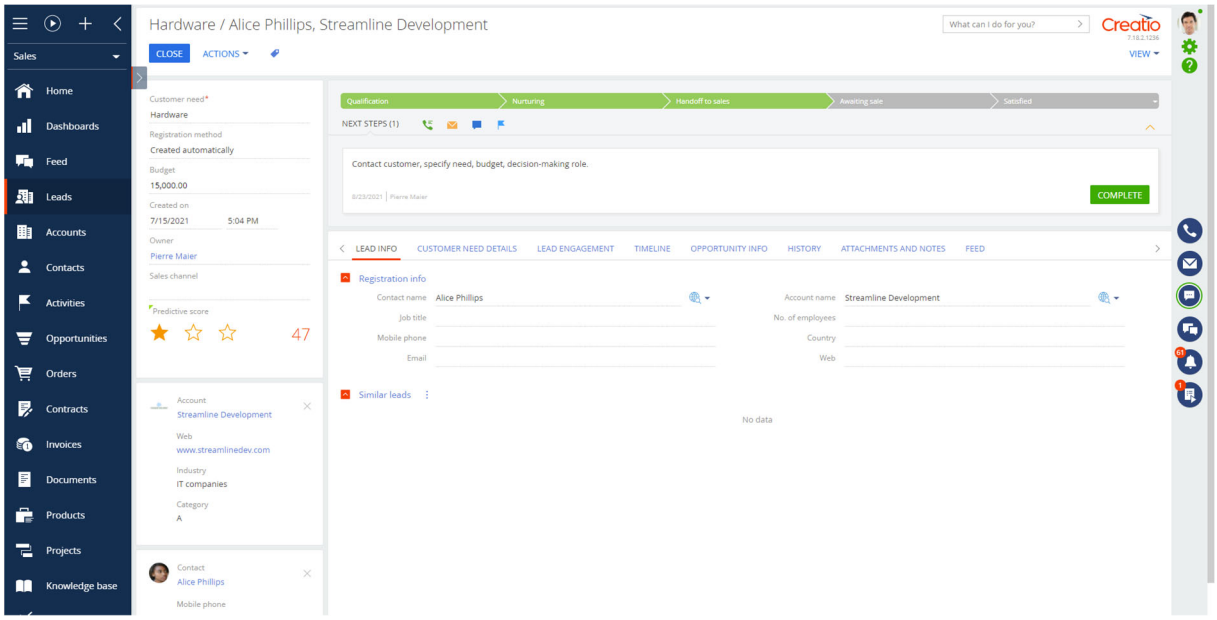

Figure 30: Creatio Sales case example

Table 30: Creatio Studio dynamic perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• business process model	• BPMN 2.0	• diagram editor • synchronized C# source code
Focus on Reuse and Adaptability		

Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• process library offers some reference workflows with generic functionalities	• BPMN 2.0 language concepts • composition	• C# code can be accessed

GUI Development. Apart from the integrated *Applications* workspace of *Creatio Studio*, additional workspaces can be added either through the Creatio marketplace or through the acquisition of further Creatio products. Other than that, it is only possible to adjust the available *Applications* workspace. The *Applications* workplace consists of a number of "sections" (see figure 24). Next to a "Home", "Dashboard", "Feed" section, each additional section represents exactly one entity type (see *Static Perspective*). Per default, a section displays all available data entries for an entity type. Every section consists of at least one "page", which allows to view and edit the available data entries. An example for a page in Creatio is displayed in figure 30. Multiple views can be designed for the same page (e.g., the elements can be arranged differently or some elements or hidden), whereby each user has always access to exactly one view of a page. Pages are developed with a drag-and-drop GUI editor. Some of the few "style" templates available on the Creatio marketplace enable a change of the color scheme or offer more icons for workspace sections, among others.

Table 31: Creatio Studio GUI development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*
• "workspace" consisting of "sections" that correspond to entity types • sections can contain multiple pages to interact with the entity type	• drag-and-drop GUI editor	GUI development can serve to simultaneously specify entity types
Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• general-purpose GUI elements are provided by Creatio • no further domain-specific elements or free-of-charge applications are available	• available GUI elements can be adjusted with regards to size and color	

Further Aspects. The cloud-based *Creatio Studio* platform can be accessed via a regular web browser. Additionally, Creatio provides an app for mobile devices running iOS or Android,

that enable access to the application use workspaces. It is possible to export entire Creatio applications in a proprietary format. The exported files can be imported only by an existing Creatio platform. The persisted data can be exported as .xlsx files.

In *Creatio Studio* two types of roles can be defined: functional roles and organizational roles. The former is assigned to a particular user, the latter defines CRUD rights for particular entity types (or, more specifically, single entities or columns). A functional role is assigned to multiple organizational roles. Organizational roles can be arranged hierarchically. This allows to aggregate specified CRUD rights, e.g., a sales department lead might have the combined rights of a sales manager and a salesperson. Collaborative use of the platform is supported through the inherent communication panel of the platform. Creatio also offers a separate *Creatio Portal*. Developers can specify restricted views on the *Applications* workspace for access through end users that are external to an organization.

For ML models, a separate module is available in the *Studio* workplace. These models can then be deployed, e.g., in workflows. ML in Creatio focuses exclusively on supervised learning. Developers can select entity types and the corresponding dependent and independent variables for a selected ML model type. The chosen entity types must be persisted on the platform. In addition to the GUI-based definition of hyperparameters, re-training can be scheduled in regular intervals to keep the productive ML model consistent with the currently available data. The inclusion of external AI services is not explicitly supported by *Creatio Studio*.

Table 32: Creatio Studio further aspects summary

Accessibility and Convenience of Use	
• no methodical development support • fairly convenient and easy-to-use • access to underlying database and source code to most user-developed artefacts • use of well-known workflow modeling language	
Modes of Use	
Platform (i.e., Application Development)	*Application*
• internal chat functionalities shall support collaborative application development • development and use of applications not clearly distinguished	• distinction between functional and organizational roles • organizational roles can be arranged hierarchically as to aggregate specified CRUD rights on higher levels • platform chat shall support collaborative application use • separate *Creatio Portal* shall enable restricted access to organization-external users
Focus on Deployment and Scalability	
Target Location	*Support*
• cloud-based platform, accessible via a regular web browser	*no further accessible support mechanisms*

Focus on Artificial Intelligence
• separate ML module enables specification and configuration of inductive, supervised ML models

5.2.2.3 Creatio Studio: Conclusion

Emphasized Areas of Application Development. Creatio Studio serves to design and develop workflow-centric applications with no dedicated support for user-friendly data management. Every application developed on the Creatio platform must follow the predefined structure of the platform with system-defined navigation and communication panel. Most features of the *Creatio Studio* LCP support the specification of functions and workflows. Workflow models are based on the well-known BPMN 2.0 specification and can include user-based tasks, conditions, and ML predictions, among others. Functions can be specified through custom C# scripts which can be embedded in workflow diagrams. A library of functions and some generic workflow models are pre-implemented on the platform.

Provision of Abstractions. Creatio aims to provide domain-specific abstractions to increase productivity. However, most provided abstractions stay on a rather generic level. Predefined entity types are scarce and cannot serve as a reference model for any domain. Available workflow models serve to provide basic functionalities to update persisted data. Nonetheless, it is possible to access more specific abstractions through the Creatio marketplace. Not all available enhancements can be used for free. Abstractions from implementation-related details are also embedded. Consider, e.g., the specification of entity types through a GUI wizard. The underlying source code for the specification of entity types, functions, and workflows can also be viewed but not edited. Interestingly, the specification of entity types and C# classes happens simultaneously through the definition of "objects". This hints at the attempt to provide business users with concepts they are familiar with, while keeping some correspondence to the underlying technology.

Role of IT professionals. Creatio explicitly advertises its platform for use by lay developers. This also conforms to our findings. Most features are not particularly demanding and do not require in-depth knowledge of programming concepts for their use. Some features, however, are hardly accessible to lay developers. This includes the specification of object inheritance relations or the writing of C#-based scripts for custom function. By this regard, professional application developers can make use of more features of *Creatio Studio* than lay developers.

5.2.3 "Low-Code" Workflow Management Systems: Conclusion

Both LCPs of this prototypical category showcase a clear emphasis in supporting the management of workflows – this applies to their marketing image as well as the platforms' features. Creatio as well as Bonita Studio have emerged around 2010 as BPM platforms and only recently adopted the low-code label to designate their platform solutions. It is apparent that not only different users but also different kinds of applications are aimed at compared to the basic

data management analyzed LCPs in subchapter 5.1. Features of the static perspective are rather faded out to some degree: Bonita's data modeling capabilities are rudimentary, and Creatio Studio does not offer any visual modeling component. Source code editing, at least in the sense of adding small scripts for enhanced functionality, is available for both LCPs. It is notable that both platforms use the term "object" simultaneously to refer not only to types and instances but also to entity types and classes.

The two vendor's marketing explicitly addresses different kinds of users: Bonita is marketed for use by professional developers and Creatio highlights use by citizen developers. The analysis of the platform features suggests any exclusive focus on developer types would be too restrictive for the respective LCP. Bonitasoft's *Form Designer* hardly demands any professional training or experience. C#-based source code editors in *Creatio Studio*, on the other hand, definitely require knowledge of object-oriented programming concepts. Both vendors thereby advertise that their LCPs shall support the collaboration and communication between business and IT professionals. No convincing features to support this purpose beyond the implemented workflow modeling features could be identified in either platform.

One key difference between the two considered LCPs lies in the development environment. *Bonita Studio* is a locally running development environment with a separate *Form Designer* module to develop applications. The development workspace of *Creatio Studio* can be accessed through a regular web browser.

5.3 Extended, GUI-, and Data-centric IDEs

LCPs within this prototypical category show some overlaps in their marketing and features to the platforms presented in the preceding two subchapters. The important difference is that no single focal point like data or workflow management is advertised. Rather, a more general support for application development can be noted. For this purpose, LCPs of this category provide LCPs that are in some parts reminiscent of common programming IDEs.

5.3.1 Mendix

Founded in the Netherlands about 15 years ago, Mendix maintains more than 15 offices in Europe, North America, Asia, and Australia. The company has recently made headlines in business newspaper, when it was acquired by Siemens, reportedly at a significant price.[38]

[38] https://www.forbes.com/sites/adrianbridgwater/2018/08/06/siemens-buys-low-code-mendix-the-digital-factory-race-climbs-higher/., accessed 09-12-2021

5.3.1.1 Mendix: Profile of Vendor

Product Portfolio. The low-code platform is the only product offered by Mendix. The LCP is separated into two development environments, one is targeted towards more experienced application developers (*Mendix Studio Pro*), while the other is advertised as proficient for citizen developers (*Mendix Studio*).

Product Provenance. The earliest archived web entry of mendix.com can be identified on Nov 30, 2005. On this date, one central slogan of Mendix' platform – then labeled *XML-based application platform* – is advertised: the platform shall support the development of "process-centric software solutions" through the provision of "off-the-shelf" software components. Since at least Dec 23, 2005, Mendix markets its platform as an approach to model-driven software development. Later, the platform is also labeled as the *Mendix Model-Driven Application Platform* on Nov 21, 2008.[39] On Dec 12, 2009, the slogan "No Code, Just Glory" is advertised. The platform is at some time marketed as an "app platform for the enterprise" (May 25, 2012) with a focus on visual and collaborative app development (Mar 01, 2014. Mendix is denoted as a low-code platform at least since May 01, 2017.

Focus on Marketing. A slogan on the Mendix homepage states: "Create better software faster by abstracting and automating the development process with Mendix, the all-in-one low-code platform."[40] On another web page, Mendix defines low-code as "a visual approach to software development" that improves cooperation between business and IT.[41] The platform is supposed to serve professional as well as citizen developers. Applications can be developed either with the "no-code", web-based *Mendix Studio* or the "low-code", locally deployed *Mendix Studio Pro* IDE. A large variety of different use cases and industries are discussed, among them smart banking[42], logistics tracking[43], and product portfolio management[44].

Table 33: Mendix profile of vendor summary

Product Portfolio
• Mendix LCP as the only product of Mendix
Product Provenance

[39] http://www.mendix.com/products (archived)

[40] http://www.mendix.com, accessed 08-26-2021

[41] https://www.mendix.com/low-code-guide/, accessed 08-26-2021

[42] https://www.mendix.com/solutions/ai-smart-banking/, accessed 08-26-2021

[43] https://www.mendix.com/solutions/iot-logistics-tracking/, accessed 08-26-2021

[44] https://www.mendix.com/solutions/product-portfolio-management/, accessed 08-26-2021

• previous emphasis on “process-centric software” and model-driven development • code avoidance in connection with application development was explicitly marketed • designated as low-code platform since 2017
Focus on Marketing
• LCP for increased development speed and improved communication between business and IT • lay and professional developers are both addressed • advertised use cases cover broad range of domains

5.3.1.2 Mendix: Analysis of Platform Features

The Mendix platform comprises nine components, five of which are directly concerned with application development.[45] Next to the already mentioned *Mendix Studio* and *Mendix Studio Pro* IDEs, these include a “data hub”, a marketplace, and *Atlas UI*. A so-called *Developer Portal* provides an overview to users of all available applications (see figure 31). *Mendix Studio Pro* is the main development IDE with a broader range of implemented features than the web-based *Mendix Studio*. The GUI of *Mendix Studio Pro* is reminiscent of a common programming IDE with a file explorer, an editor window, and a console.

[45] The remaining components provide rather indirect support for application development, such as a forum for developers, a Mendix FAQ, or an online academy for platform training.

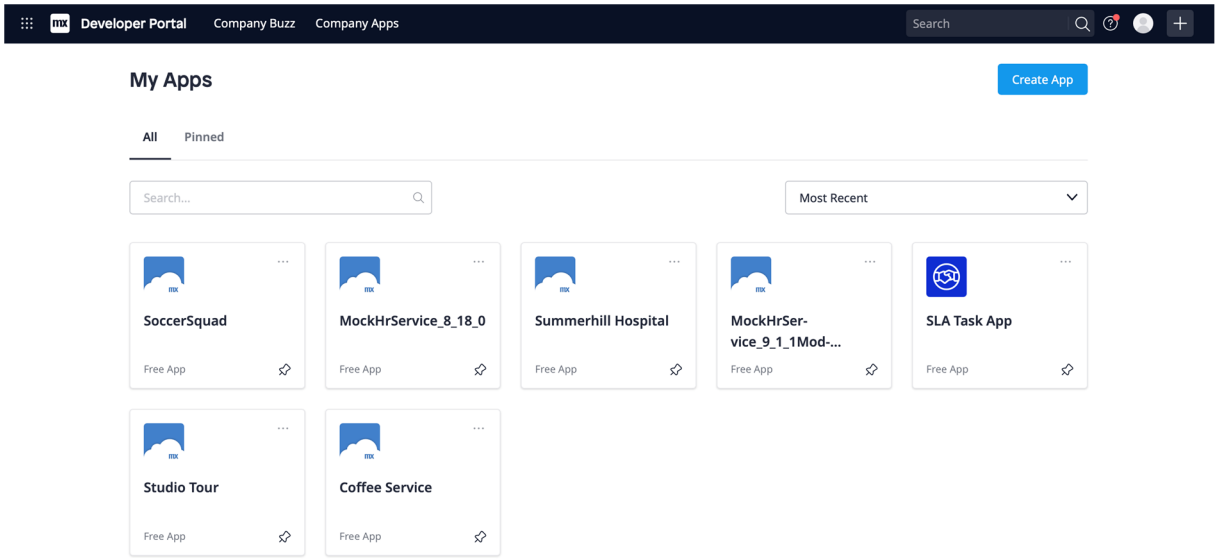

Figure 31: Mendix Developer Portal

Static Perspective. Every application consists of two system-generated "modules" – a system module encompasses all Mendix-provided functionalities, a custom module can be used by developers to specify application artefacts. Developers can also add more modules. The modules of an application can be accessed in *Mendix Studio Pro.* Every module contains exactly one "domain model", a set of GUI pages, and further files (e.g., microflows, JSON structures, or Java actions, see subsequent analysis sections). A "domain model" is a data model. An example is displayed in figure 32. Entity types can be defined through common data types such as Integer or String. No domain-specific data or entity types are pre-implemented. Although complete Mendix applications can be downloaded on the Mendix marketplace, reference data models are not available. Domain models follow an ERM-like notation. Developers can specify one-to-one, one-to-many, and many-to-many associations between entity types. It is possible to refer to entity types from other domain models within the same application (see the "Employee_Account" association in figure 32). Inheritance of entity types is also supported. In figure 32, the "Employee" entity type inherits from the "System.User" entity type. The specification of entity types allows to auto-generate corresponding Java classes. The object-relational mapping is inaccessible and cannot be adjusted. Data can be persisted by Mendix. To use externally persisted data, a mapping between internal and external entity types must be defined (see figure 33). It is also possible to specify transient entity types, which are displayed as orange boxes in a domain model (see figure 32). Integration of data schemas between Mendix applications shall be supported through Mendix' "data hub" but this component requires an additional license and is thus not considered here.

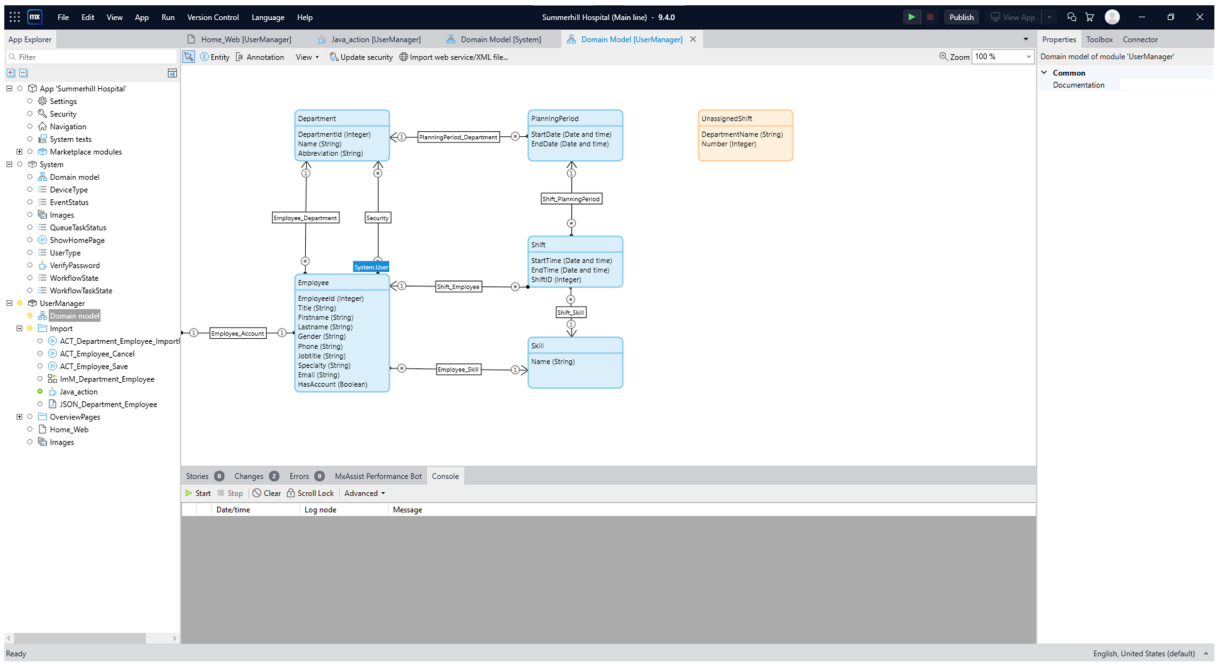

Figure 32: Mendix Studio Pro domain model

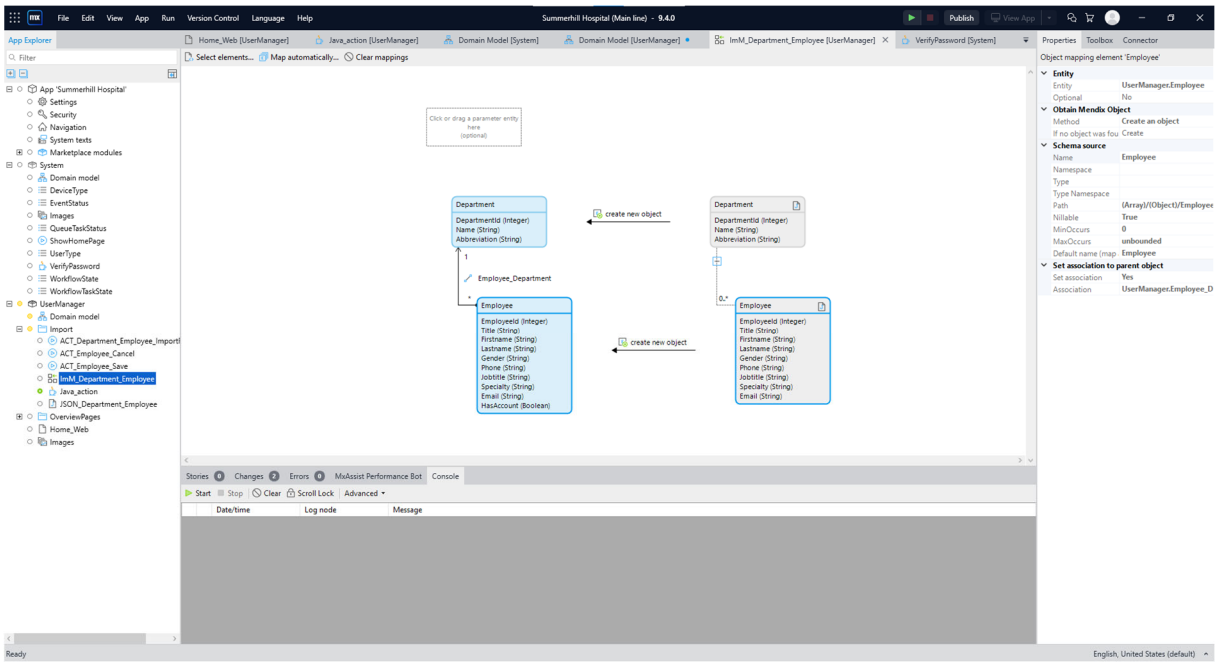

Figure 33: Mendix Studio Pro data import mapping

Table 34: Mendix static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• data model (“domain model”)	• ERM-like notation	• model editor • Java code generator

Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• only generic data types are offered • domain-specific reference abstractions might be accessed through the Mendix marketplace	• one-to-one, one-to-many, many-to-many associations • inheritance of entity types • Distinction between transient/persistent data	*Implementation-level documents cannot be accessed within the development environment*
Focus on Integration		
Common static abstractions across a range of applications	use of common static abstractions depends on developer; is generally supported	
Access to common data repositories across a range of applications	depends on developer, generally possible to, inter alia, persist all data within Mendix for this purpose	

Functional Perspective. Some generic functionalities are pre-implemented for use in micro- or nanoflows (see *Dynamic Perspective*), e.g., the generation of documents or showing web pages. Rules and expressions can also be modeled via simple conditions and actions (see figure 34). Additional functionalities beyond the offered tasks for rule and workflow modeling need to be implemented using Java or JavaScript. The definition of Java or JavaScript functions within *Mendix Studio Pro* focusses on a contract-like definition of parameter and return types, not the actual implementation. Source code editing within *Mendix Studio Pro* is only possible for JavaScript (see figure 35). Java functions must be implemented in another IDE, no built-in Java source code editor is provided. Mendix offers a set of APIs and an SDK for developers to interact with a Mendix application.[46] Support for the integration of external functions is provided in the workflow modeling components (see *Dynamic Perspective*).

[46] See https://docs.mendix.com/apidocs-mxsdk/ for a full list. Accessed 09-01-2021

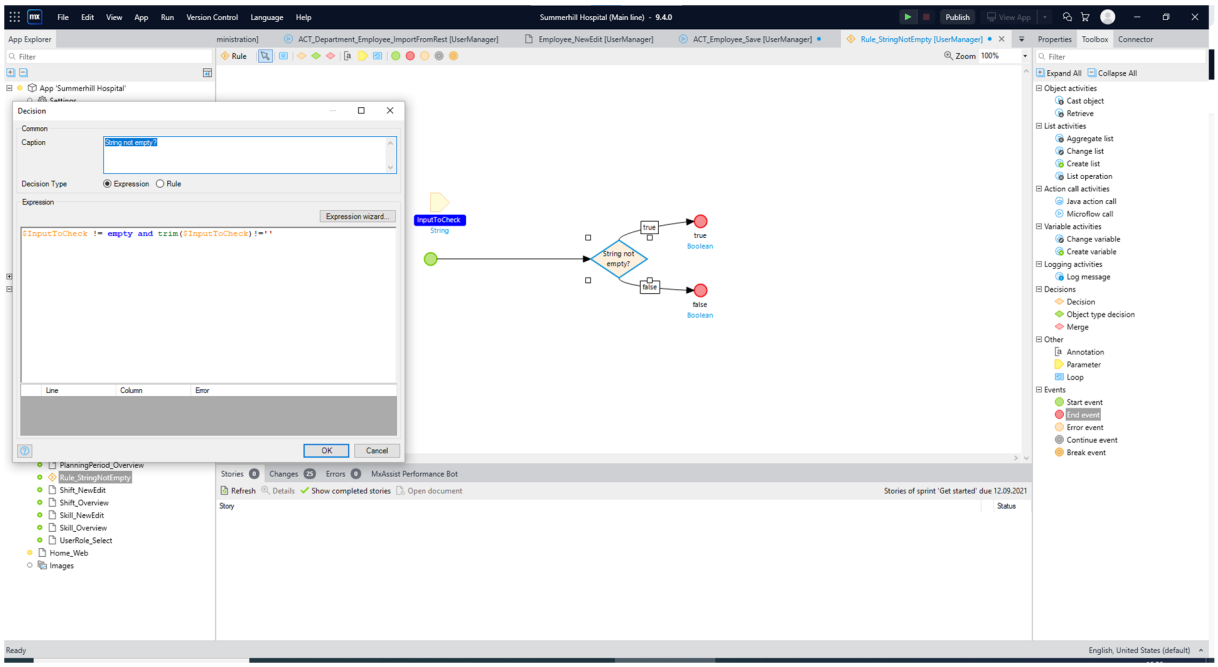

Figure 34: Mendix Studio Pro validation rule modeling and expression definition

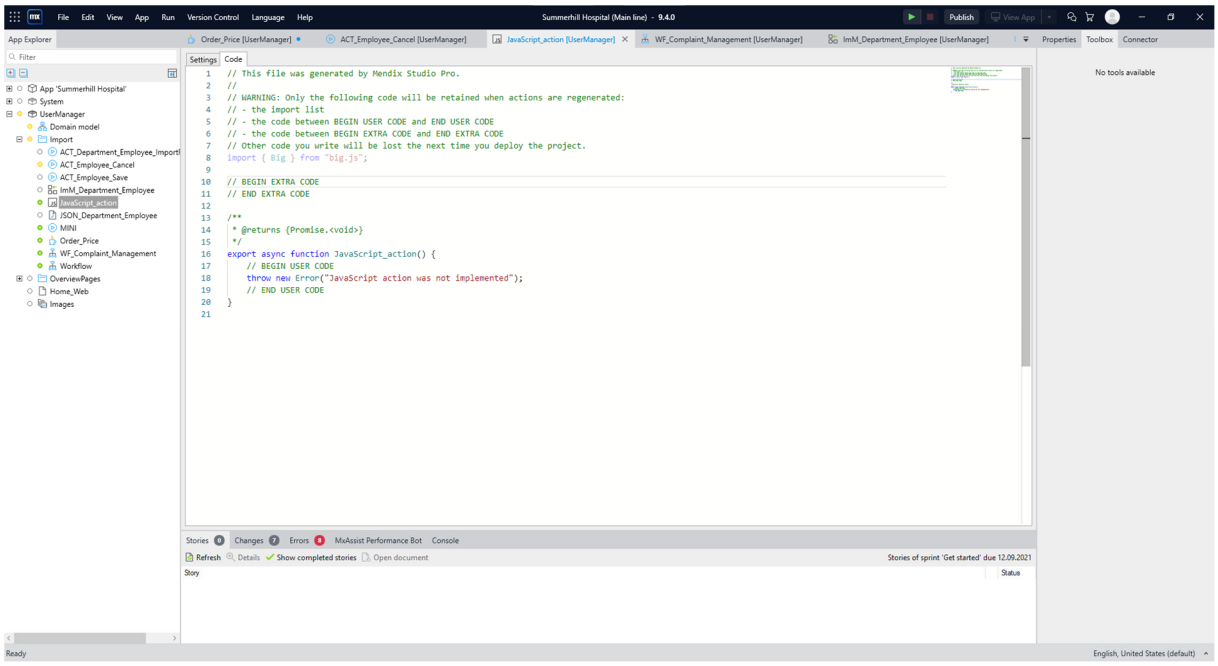

Figure 35: Mendix Studio Pro JavaScript source code editing

Table 35: Mendix functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*

• rule models	• proprietary	• diagram editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic abstractions provided like conditionals or activities	• composition	• access to libraries and SDK • built-in JavaScript source code editor • support for integration of external sources in workflow models

Dynamic Perspective. Three different workflow model types are provided in *Mendix Studio Pro*: "nanoflows", "microflows", and "workflows". Conceptually, nanoflows and microflows are identical, i.e., they implement the same modeling components consisting of a large set of pre-implemented "activities", "decisions", and "events" (see figure 36). The difference between these two workflow model types lies in implementation-related details. For example, nanoflows are executed client-side (potentially relevant for native, offline running mobile apps) while microflows are executed server-side. Nanoflows and microflows focus exclusively on automated tasks (i.e., where no user input is required), while "workflows" (currently still in a beta-version) offer a more restricted set of modeling concepts only to combine microflows with manual interactions from users. An example of such a "workflow" is displayed in figure 37. Nano- and microflows can serve to define rule-based functionalities that are executed at certain predefined events (e.g., button on-click). This can include calling REST services, data imports/exports, or some GUI-based operations (e.g., open page).

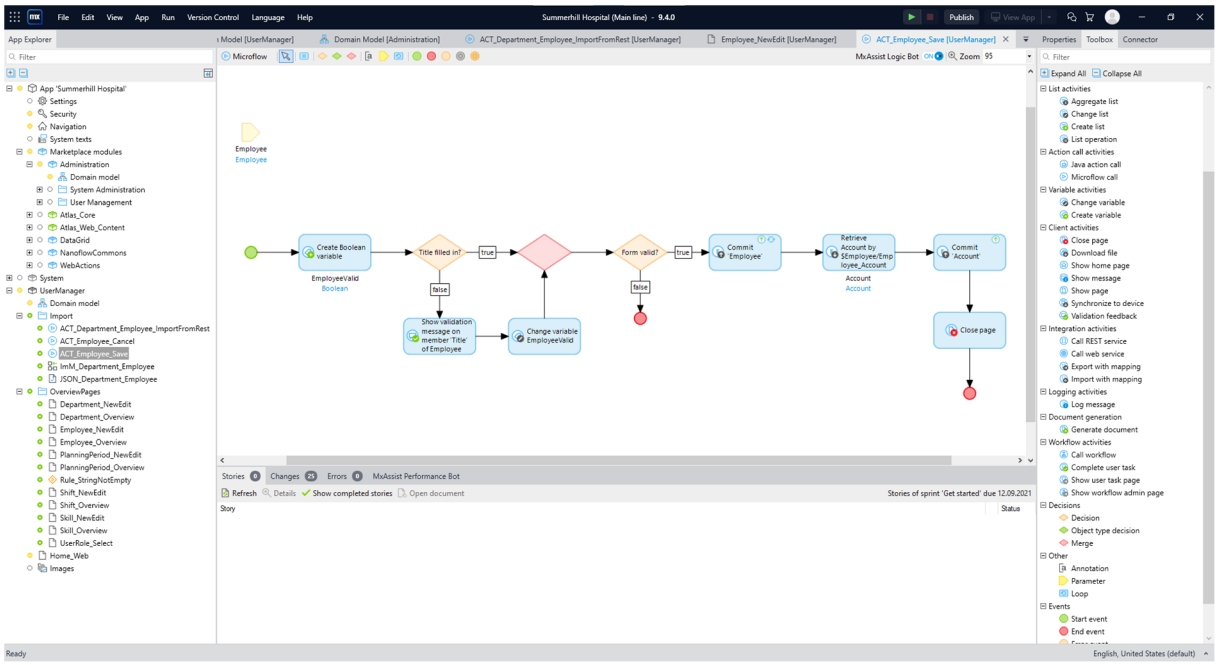

Figure 36: Mendix Studio Pro microflow

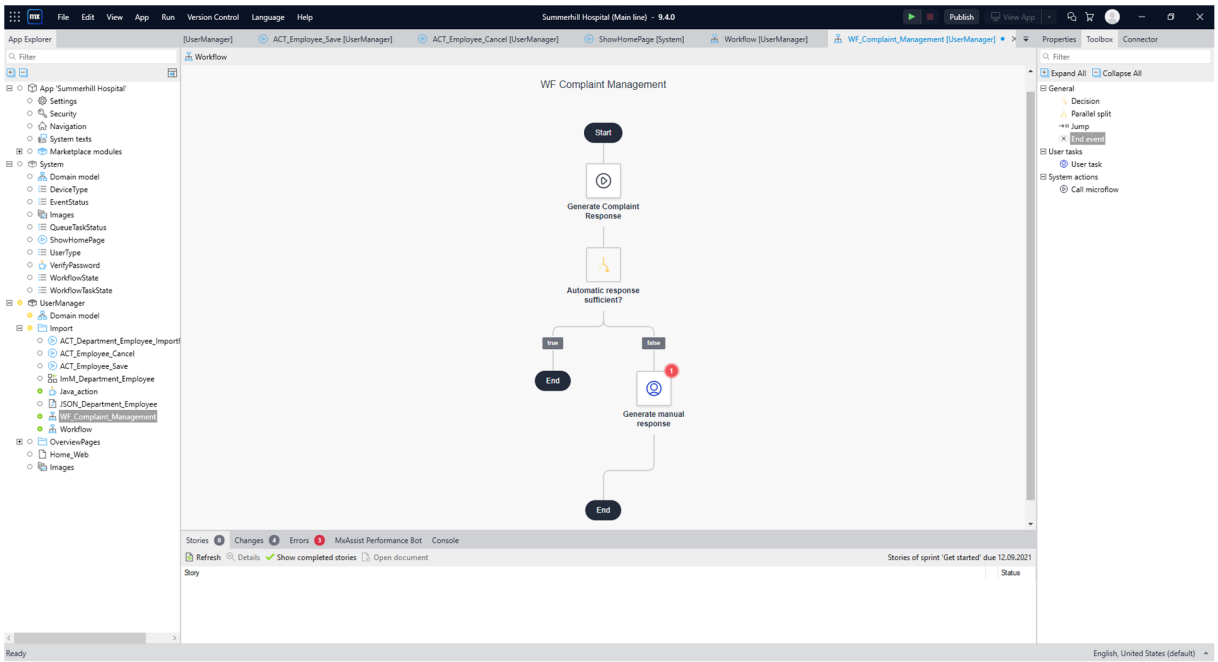

Figure 37: Mendix Studio Pro "workflow"

Table 36: Mendix dynamic perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• generic process models	• based on BPMN (nanoflows, microflows)	• model editor

	• proprietary ("work-flows")	
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic abstractions like events or activities	• composition	*no access to implementation-level documents*

GUI Development. GUI pages can be created automatically based on a defined domain model. For each entity type an "overview" and "edit" page is created per default, respectively showing persisted data entries and allowing to add a new data entry. An example for an edit page in *Mendix Studio Pro* is displayed in figure 38. Next to the single pages for viewing and adding data entries, each application contains one navigation panel (see figure 39). *Atlas UI* is a collection of libraries for different generic layout and GUI elements. Other libraries may be self-developed and integrated within the Mendix environment. A number of "app templates" serve as reference GUIs (see figure 40). Per default, each application is based on a responsive web GUI, which offers layout elements that adjust automatically to different screen sizes. Each GUI page can be designed with a drag-and-drop GUI editor. GUI pages can also be edited in *Mendix Studio* (see figure 41).

Table 37: Mendix GUI development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*
• GUI model	• drag-and-drop GUI Editor	implementation of MVC pattern
Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• library of general-purpose GUI elements through *Atlas UI*	• GUI elements can be adjusted with regards to their position, color, and size	

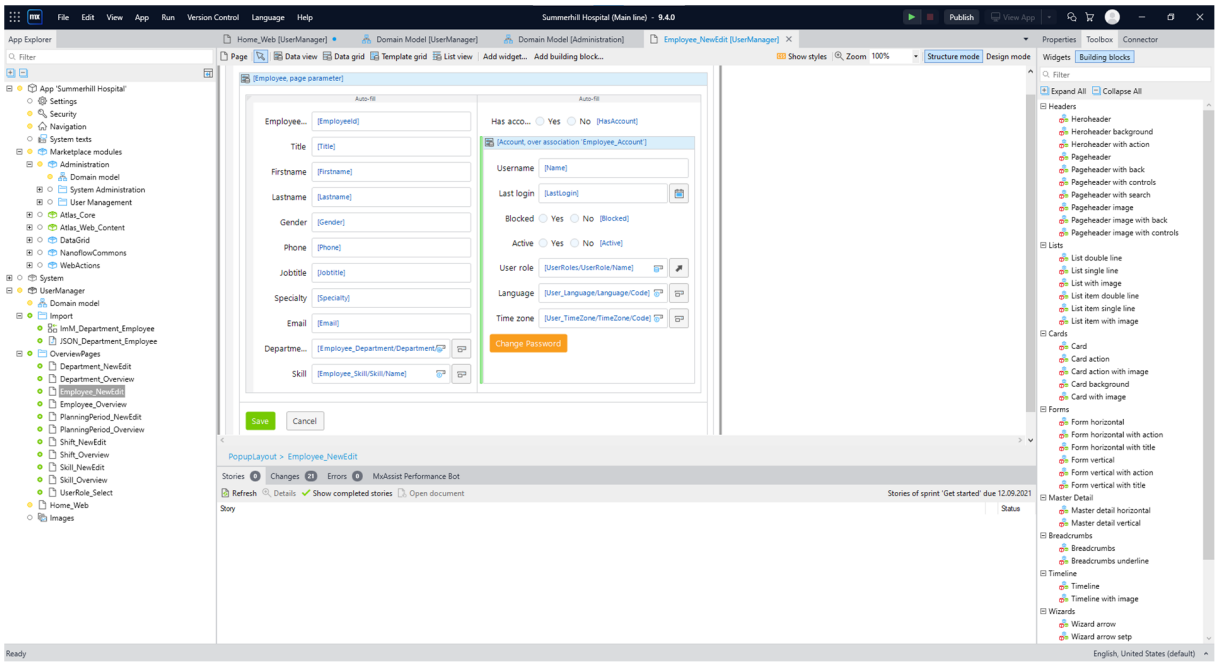

Figure 38: Mendix Studio Pro new edit GUI page editor

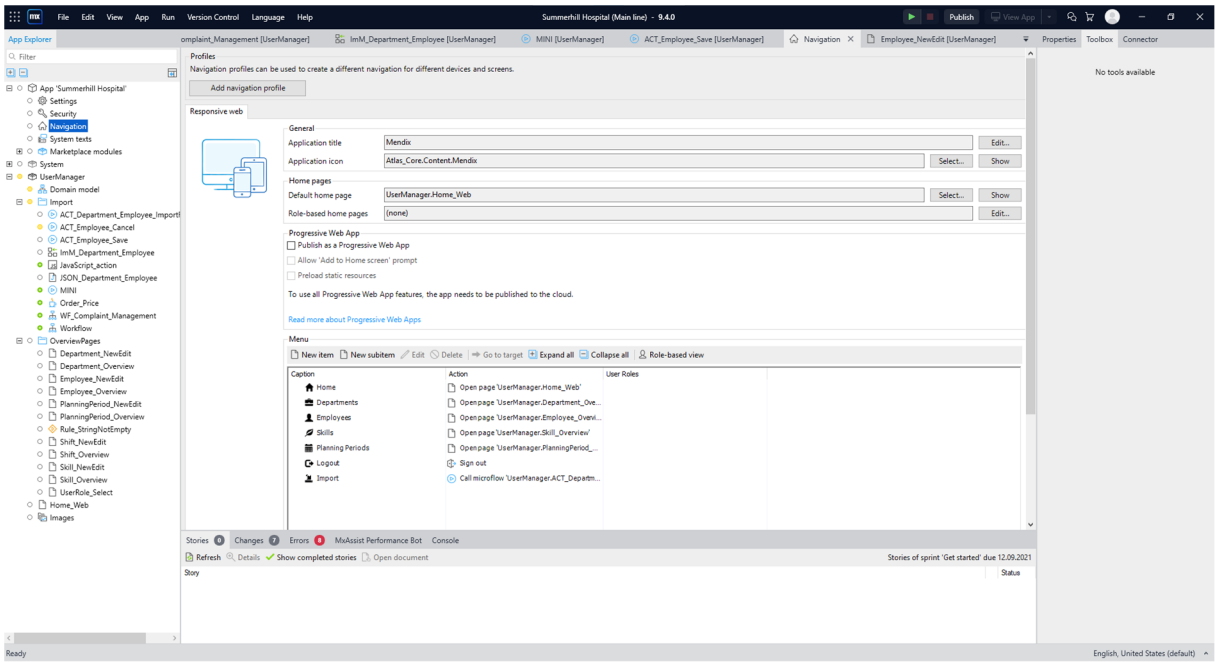

Figure 39: Mendix Studio Pro navigation page editor

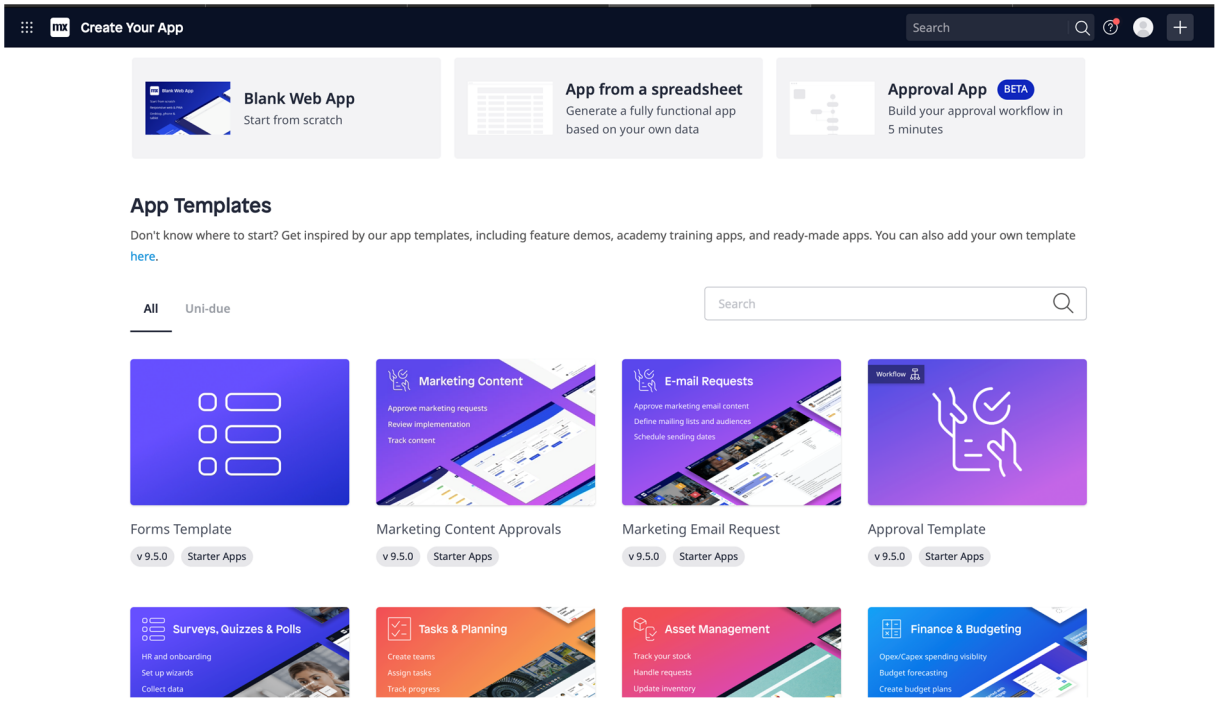

Figure 40: Mendix Studio app templates

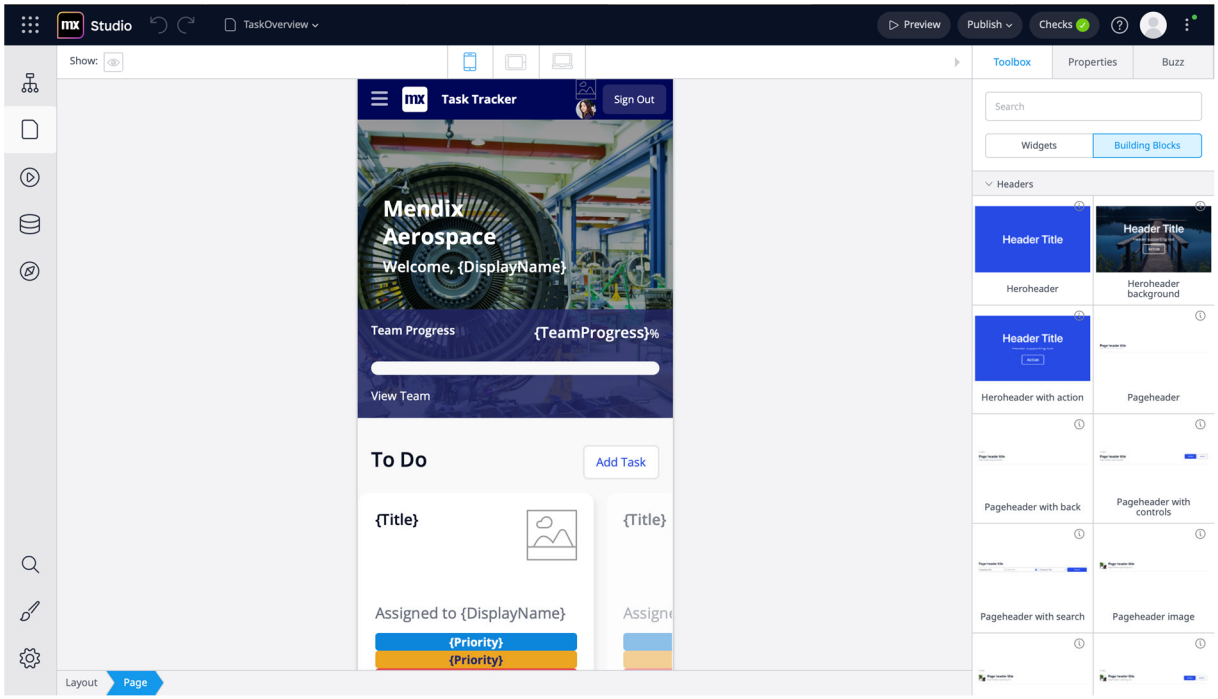

Figure 41: Mendix Studio GUI editor

Further Aspects. Mendix applications are deployed on a cloud server and can be accessed via a custom URL through a common web browser. The locally deployed *Mendix Studio Pro* comes with an integrated version management system. Mendix provides some features that shall support agile software development methods, e.g., to list user stories, show sprint dates, or update the product backlog. User stories can be accessed *Mendix Studio* and *Mendix Studio Pro*. Within *Mendix Studio*, a "Buzz" tab is available for each GUI page, where users can exchange comments in reference to particular GUI elements. These comments can be viewed in the *Developer Portal*.

User and module roles can be specified in *Mendix Studio Pro*. Each user is assigned to a user role and each user role can be assigned to multiple module roles. Module roles can be provided with unrestricted access rights to GUI pages, various workflow model types, and domain models. A more specific differentiation in CRUD rights is not offered.

Mendix Studio Pro offers an *Mx Assist Logic Bot* which is considered AI. The "bot" provides "next best action" advice when designing micro- and nanoflows. After each node in the diagram, a number of possible actions are suggested that might follow the preceding node. It is unclear how the ML model to achieve this has been trained, or what ML model is even underlying this mechanism. This makes the bot's effectiveness not assessable and the results appear as more or less arbitrary suggestions.

Mendix runs a marketplace for software artefacts such as "app services", "templates", "widgets", etc. It is supposed to attract contributions from third parties in order to generate considerable value for Mendix users. Currently, the offer is modest in scope and largely restricted to contributions by Mendix itself.

Table 38: Mendix further aspects summary

<table>
<tr><td colspan="2">Accessibility and Convenience of Use</td></tr>
<tr><td colspan="2">• two IDEs (Mendix Studio and Mendix Studio Pro) address different kinds of developers
  ○ Mendix Studio is fairly convenient and intuitive
  ○ Mendix Studio Pro demands some training and knowledge of development-related concepts
• methodical support shall be provided through the inclusion of Scrum-related concepts like user stories, product backlog, and others
• use of multiple models for data and different workflows that are based on standard modeling notations</td></tr>
<tr><td colspan="2">Modes of Use</td></tr>
<tr><td>Platform (i.e., Application Development)</td><td>Application</td></tr>
<tr><td>• locally deployed IDE that runs version control system
• access rights managed through “module roles”</td><td>• access rights managed through “module roles”
• no inherent support for collaborative use of applications</td></tr>
<tr><td colspan="2">Focus on Deployment and Scalability</td></tr>
<tr><td>Target Location</td><td>Support</td></tr>
<tr><td>• Mendix public cloud server per default, application can be accessed via web browser
• alternative deployment options (e.g., private cloud) are also supported</td><td>• use of Kubernetes for container orchestration</td></tr>
<tr><td colspan="2">Focus on Artificial Intelligence</td></tr>
<tr><td colspan="2">• Mx Assist Logic Bot to support modeling of workflows not convincing</td></tr>
</table>

5.3.1.3 Mendix: Conclusion

Emphasized Areas of Application Development. Mendix provides two development environments that emphasize different aspects of application development. *Mendix Studio* allows users to intuitively design web-based GUIs. *Mendix Studio Pro* enables developers to define multiple data and workflow models. Additionally, the Mendix environment provides methodical support for agile application development. This is showcased by the possibility to define user stories, sprints, and a product backlog. No single area of application development is exclusively addressed.

Provision of Abstractions. *Mendix Studio* and *Mendix Studio Pro* fade out implementation-related details to a larger extent. In *Mendix Studio*, the GUI can only be edited via drag-and-drop of predefined GUI elements. In *Mendix Studio Pro*, a broad range of visual models enable development without specifying source code. Domain-specific abstractions are only partially addressed through the available GUI templates and the Mendix marketplace. A lack of domain-specific reference models can also be noted in *Mendix Studio Pro*.

Role of IT Professionals. The two development environments of Mendix clearly address different kinds of application developers. Lay developers can use *Mendix Studio* to specify application

GUIs. Users of *Mendix Studio Pro* have access to a broader range of features. Professional developers are concerned with the specification of most static, functional, and dynamic aspects of an application while lay developers can adjust GUI pages, add comments to GUI elements, and manage user stories. Integration between both views is not aimed to be achieved via a conceptual integration of common development concepts (like class, database, workflow, etc.), but rather through some additional communication mechanisms between both development environments.

5.3.2 WaveMaker

The company behind WaveMaker, also named WaveMaker, seems to have its roots in an Indian software company. In 2013, it was acquired by VMWare, a large US software firm that is focused on software virtualization.

5.3.2.1 WaveMaker: Profile of Vendor

Product Portfolio. The WaveMaker LCP is the only product offered by WaveMaker.

Product Provenance. The company is called WaveMaker only since 2007. Previously, it was called *ActiveGrid*. The earliest archived web page of ActiveGrid can be identified on Jul 29, 2004. The ActiveGrid platform is advertised to consist of two components: a "grid application server" and an "application builder". The former is associated features around scalability and deployment (Mar 30, 2005)[47], the latter is described as a "rapid application development environment […] which enables to graphically and iteratively create, deploy, and test enterprise applications" (Apr 09, 2005)[48]. The rebranding of ActiveGrid to WaveMaker in 2007 did not result in decisive differences in the product offering. WaveMaker claims to enable "everyone [to] quickly build and deploy great-looking web applications" with a "visual AJAX" locally running IDE (Sep 30, 2008). Application development should be faster and require close to no coding (Dec 01, 2008). The platform does not carry any particular label, but customer testimonials like "It took me less than 10 minutes to build an Employee maintenance application. And I'm not even a developer!" (Oct 03, 2011) correspond to current low-code promises. WaveMaker itself states that its platform has been rebuilt as an LCP since 2013[49], but the analysis of its archived web presence suggests otherwise. The relabeling to a low-code platform can be noted at least since Feb 02, 2017, which does not seem to be accompanied by particular technological changes.

[47] http://www.activegrid.com/products/index.php (archived)

[48] http://www.activegrid.com/products/ab.php (archived)

[49] https://www.wavemaker.com/about/, accessed 08-30-2021

Focus on Marketing. Low-code development on the WaveMaker platform is understood as a mixture of RAD, aPaaS, BPM, and no-code.[50] In reference to these concepts, the vendor promotes low-code as offering (i) techniques to increase development speed (RAD), (ii) a single cloud-based platform for development and deployment (PaaS), (iii) automation and modeling of business processes (BPM), and (iv) a drag-and-drop GUI editor (no-code). The company explicitly targets professional developers in their marketing messages and promotes improved alignment of business and IT.

Table 39: WaveMaker profile of vendor summary

Product Portfolio
• LCP is the only product of WaveMaker
Product Provenance
• early distinction between application server and rapid application development • code avoidance and broad accessibility is advertised since at least 2009 • labelled as LCP since early 2017
Focus on Marketing
• improved business-it-alignment • quicker application development • business process automation • emphasis on professional application developers

5.3.2.2 WaveMaker: Analysis of Platform Features

The WaveMaker platform consists of five modules for development: "Pages", "Databases", "Web Services", "Java Services", and "APIs". The home screen of the WaveMaker platform, which provides an overview of all available apps, is displayed in figure 42.

[50] https://www.wavemaker.com/application-platform-as-a-service-apaas/, accessed 09-01-2021

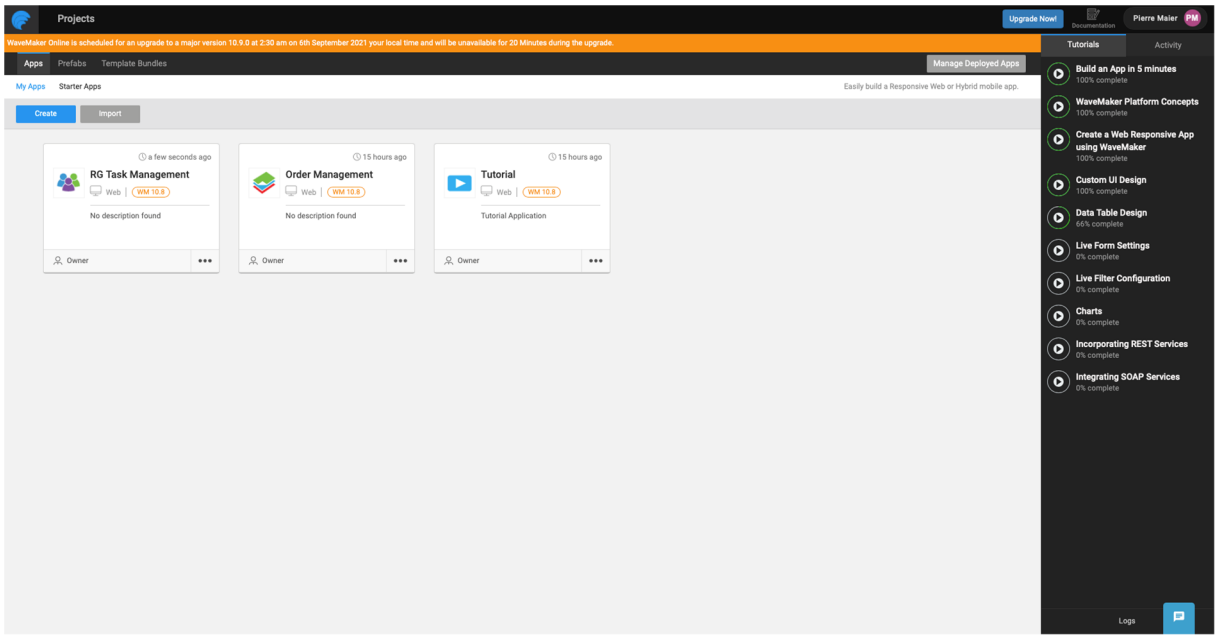

Figure 42: WaveMaker platform home screen

Static Perspective. Features of the static perspective are mainly provided through the "Database" module of the WaveMaker platform. It is possible to specify a connection to an external database and access/edit its schema within WaveMaker. Data can also be persisted on the WaveMaker platform. Data models are based on a proprietary modeling language, loosely based on an ERM-like notation (see figure 43). The definition of entity types and associations is close to common database definitions: data types for attributes are based on SQL and Java (i.e., the mapping of data types is accessible). Their specification can be refined by familiar constraints. (e.g., nullable, unique). Many-to-many associations need to be defined through an auxiliary table, each table requires at least one primary key. Each entity type corresponds to one Java class. The exact object-relational mapping is not accessible. Adjustments to the data model result in an SQL statement that can be accessed and modified before execution. A separate "Query" tab provides developers with a built-in editor to define HQL (Hibernate Query Language) and SQL queries. Reference data models are not available.

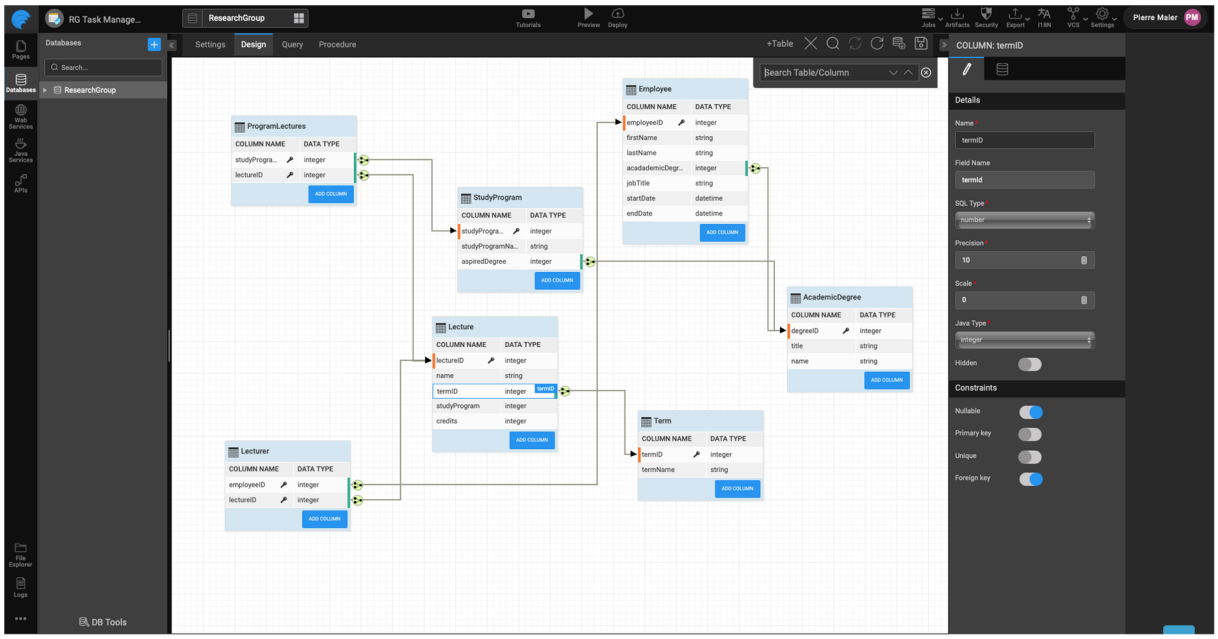

Figure 43: WaveMaker data model

Table 40: WaveMaker static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*

• data model	• proprietary	• model editor
		• generator
		  ○ SQL statements for relational database
		  ○ corresponding Java classes

Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• only generic data types like string, int, etc. are available	• generalization • encapsulation • composition • …	• auto-generated SQL (DDL/DML) statements can be accessed and adjusted • built-in HQL and SQL editor can be used for further queries
Focus on Integration		
Common static abstractions across a range of applications	No direct support for common static abstractions is offered. The developer is responsible for this.	
Access to common data repositories across a range of applications	All data persisted by WaveMaker is in the same data repository.	

Functional Perspective. WaveMaker provides developers with GUI forms to guide the integration of web services through REST, SOAP, and WebSockets. Custom "Java Services" can be defined with a built-in source code editor (see figure 44). WaveMaker offers several runtime APIs for use in these Java Services. To assign a function to GUI elements, additional REST APIs must be specified. These are generated automatically for user-defined Java Services. Some REST APIs with the corresponding functionality are pre-implemented, such as, e.g., CRUD APIs to access database entities.

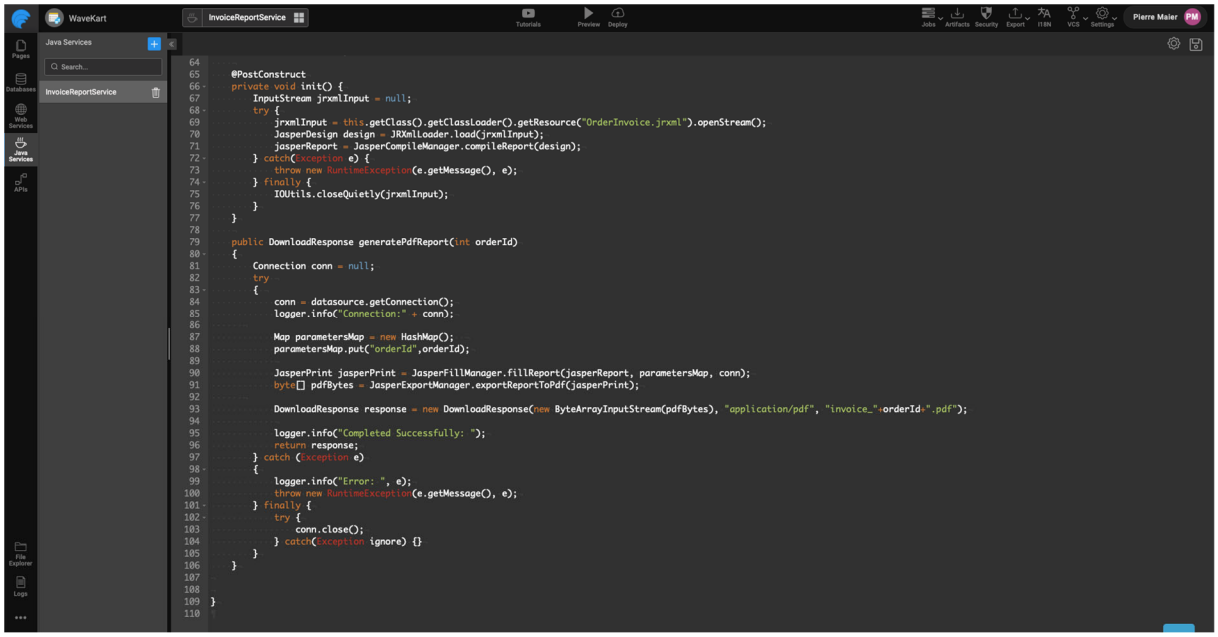

Figure 44: WaveMaker Studio Java source code editor

Table 41: WaveMaker functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• only represented as Java functions	• Java	• built-in code editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• no particular abstractions beyond Java source code	• no particular language concepts beyond Java concepts	• Java source code can be accessed and edited • support for the integration of external sources via REST, SOAP, and WebSockets • provision of runtime APIs

Dynamic Perspective. WaveMaker does not offer any explicit support for workflows. No workflow modeling component is available. Dynamic aspects are largely defined through a user's interaction with GUI pages. For example, CRUD event listeners for a database can be accessed via a runtime API in Java services, and other form-based event listeners can be assigned to "variables" in the GUI editor (see *GUI Development*).

GUI Development. Each WaveMaker application consist of a number of GUI screens ("Pages"), which can be designed through a visual drag-and-drop interface (see figure 46). The JavaScript , HTML and CSS code can be accessed and edited through built-in code editors. A GUI page can be created based on a pre-defined template and a corresponding theme that determines the color scheme of a page. Chosen templates can be edited and it is also possible to define templates and themes independent of any particular application for reuse across WaveMaker applications. Pages can contain widgets and so-called "prefabs". Widgets range from dialog windows to charts, buttons, and simple containers. "Variables" in WaveMaker Studio are the binding element between the "model" and the "view" in the implemented MVC pattern. They define some target action like, e.g., retrieving data from a table or executing a Java operation. Any functional GUI element needs to be manually connected with a variable. No automatic GUI generation based on data models is available. Prefabs denote some GUI container with specifically defined actions as, e.g., an Instagram Login prefab or a Google Maps prefabs. Eight prefabs are currently pre-implemented within WaveMaker Studio.

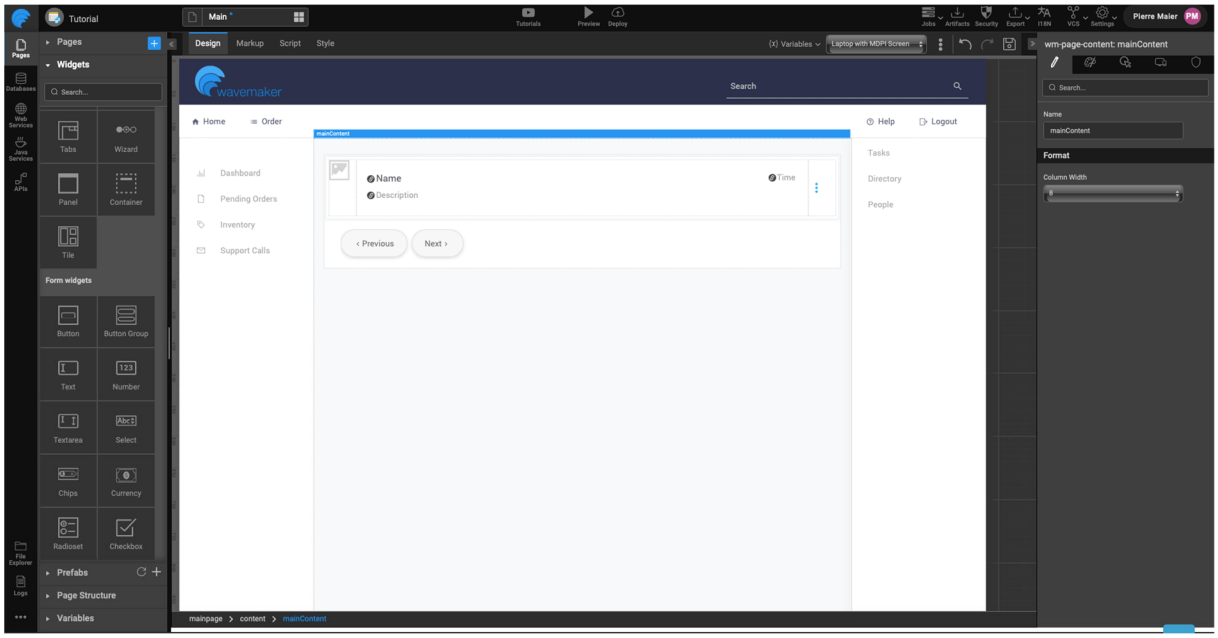

Figure 45: WaveMaker Studio GUI designer

Table 42: WaveMaker GUI Development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*

• GUI model	• GUI Editor
	  ○ drag-and-drop
	  ○ WYSIWYG
	• Synchronized markup (HTML), style (CSS), and script (JavaScript) editors
		implementation of MVC pattern

Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• library of general-purpose GUI elements	• all GUI elements can be adjusted according to their size, position, and style	

Further Aspects. The WaveMaker platform can be accessed via a web browser and responsive web applications can be deployed on WaveMaker or other web servers. The entire source code and associated files for an application can be exported and accessed via other IDEs. It is, however, not possible to access WaveMaker-specific libraries.

WaveMaker Studio provides a version control system to support collaborative development. The exact mechanisms of this version control system are inaccessible. The definition of separate end user and developer roles is not explicitly supported by the platform. More extensive licensing models of the platform are said to include a "Team collaboration" module for managing different users and roles.[51] The current analysis relies on the solo version of WaveMaker Studio, omitting any detailed assessment of this feature.

No specific AI components are part of the WaveMaker platform. It is, however, feasible to develop AI functions with Java or to integrate AI services via a generic API.

[51] https://docs.wavemaker.com/learn/teams/overview, accessed 09-03-2021

Table 43: WaveMaker further aspects summary

Accessibility and Convenience of Use	
• no methodical application development support • except for superficial data model adjustment and GUI development, the platform largely requires some sort of source code specification • proprietary data modeling language	
Modes of Use	
Platform (i.e., Application Development)	*Application*
• version control to support collaborative development • elaborate definition of developer roles not assessable in considered version	• no direct support for collaborative application use by WaveMaker
Focus on Deployment and Scalability	
Target Location	*Support*
• cloud-based platform, accessible via a regular web browser • domain name can be customized for applications	• support to build docker images
Focus on Artificial Intelligence	
• no specific support for any AI service	

5.3.2.3 WaveMaker: Conclusion

Emphasized Areas of Application Development. The core features of the WaveMaker platform are its drag-and-drop GUI editor, a data modeling component, and the specification of custom Java operations. Built-in source code editors are of high prominence in WaveMaker and can be accessed for each of the mentioned components. WaveMaker can be described as a web-based IDE with a built-in version management system and additional support for GUI development, data modeling, and deployment.

Provision of Abstractions. Productivity increase is achieved through the provision of additional representations of source code. This includes the capacity for visual data modeling and the provision of visual GUI elements. For these cases the corresponding source code can also be accessed, enabling the respective developer to choose between the provided representations. Domain-specific abstractions are not provided by WaveMaker. However, it would be possible for a developer to specify domain-specific GUI artefacts that can be accessed across all WaveMaker application.

Role of IT Professionals. WaveMaker explicitly addresses only professional developers in their marketing. The analysis of platform features largely supports this claim. Source code editors are of high prominence and the specification of custom functions must be code-based. The

features of the WaveMaker platform reduce the effort for recurring tasks of application development, such as the implementation of the MVC pattern, an object-relational mapping, or the development of GUI pages. Lay developers might use the LCP only superficially through visual adjustment of visual data models or single GUI pages. Collaboration between differently trained application developers is not addressed.

5.3.3 Zoho Creator

Zoho, originally named *AdventNet*, the vendor of the Zoho Creator platform, was founded in India about 25 years ago. According to its web pages, it has grown into a company that runs offices on all continents.[52]

5.3.3.1 Zoho Creator: Appearance of Vendor

Product Portfolio. Zoho currently offers approximately eighty different products that are categorized according to different needs (e.g., sales, legal, or collaboration) and can be accessed in various bundles (e.g., CRM or IT management). The entire catalog of products is aggregated in the Zoho One platform, which is marketed as the "Operating System for Business".[53] Two of these eighty products are labeled "no-code" and one is designated as a low-code platform. The LCP of Zoho is called *Zoho Creator*

Product Provenance. Zoho's original parent company AdventNet was founded in 1996.[54] Zoho Creator was among the first products the company released and is publicly available since March 2006. AdventNet was renamed to Zoho Corporation in 2009. Since its early advent, the platform has advertised increased development speed with "no coding required" (Sep 02, 2006). On March 21, 2007 the development of "online database applications" is advertised. The following years alternate between the notions of "database applications" and "business application" until the terminological focus on databases is, apparently, dropped in late 2016. Since at least Aug 30, 2009, workflows and business rules are advertised as features of the platform. Additional claims, like enabling business users to develop applications (Aug 03, 2015), show clear parallels to common promises associated with low-code. On October 08, 2015, the platform is described as an "easy-to-use and low coding platform". It is designated as an LCP since at least Jun 28, 2020. No particular changes in the platform's features or promises can be noted.

Focus on Marketing. Zoho advertises generic benefits of LCPs: applications can be developed quicker and costs can be reduced. [55] To achieve this, "prebuilt app-building components" and

---

[52] https://www.zoho.com/contactus.html, accessed 09-02-2021

[53] https://www.zoho.com/one/overview.html, accessed 09-02-2021

[54] https://www.zoho.com/aboutus.html, accessed 09-02-2021 (see also for a complete list of product introductions)

[55] https://www.zoho.com/creator/overview.html, accessed 09-05-2021

an "easy to understand logical scripting language" are mentioned.[56] Both, citizen and professional developers are addressed in Zoho's marketing statements. Advertised use cases for Zoho Creator include enterprise architecture, legacy modernization, business process management software, and workflow automation

Table 44: Zoho Creator profile of vendor summary

Product Portfolio
• broad range of approximately 80 software products as part of "Zoho One" landscape • LCP as one of four application development products
Product Provenance
• early focus on code avoidance can be noted since 2006 • focus shifted from "database applications" to "business applications" • labelled as LCP since 2020
Focus on Marketing
• quicker development and reduced costs through reusable app components • focus on lay and professional developers" • broad range of advertised use cases

5.3.3.2 Zoho Creator: Analysis of Platform Features

The home screen of the Zoho Creator platform provides an overview of all accessible applications (see figure 46). An application in Zoho Creator consists of "pages", "forms", "reports", and "workflows". Pages, forms, and reports all denote some type of GUI page (see *GUI Development*). All applications consist of two GUI elements: a navigation bar and the set of implemented GUI pages. It is also relevant to distinguish two available development environments within Zoho Creator. At first, there is a more restrictive and user-friendly editor which is based solely on drag-and-drop interfaces and display forms. At second, access to source code and further functionalities is provided through the respective "developer tools" of an application.

[56] https://www.zoho.com/creator/application-development/low-code.html, accessed 09-05-2021

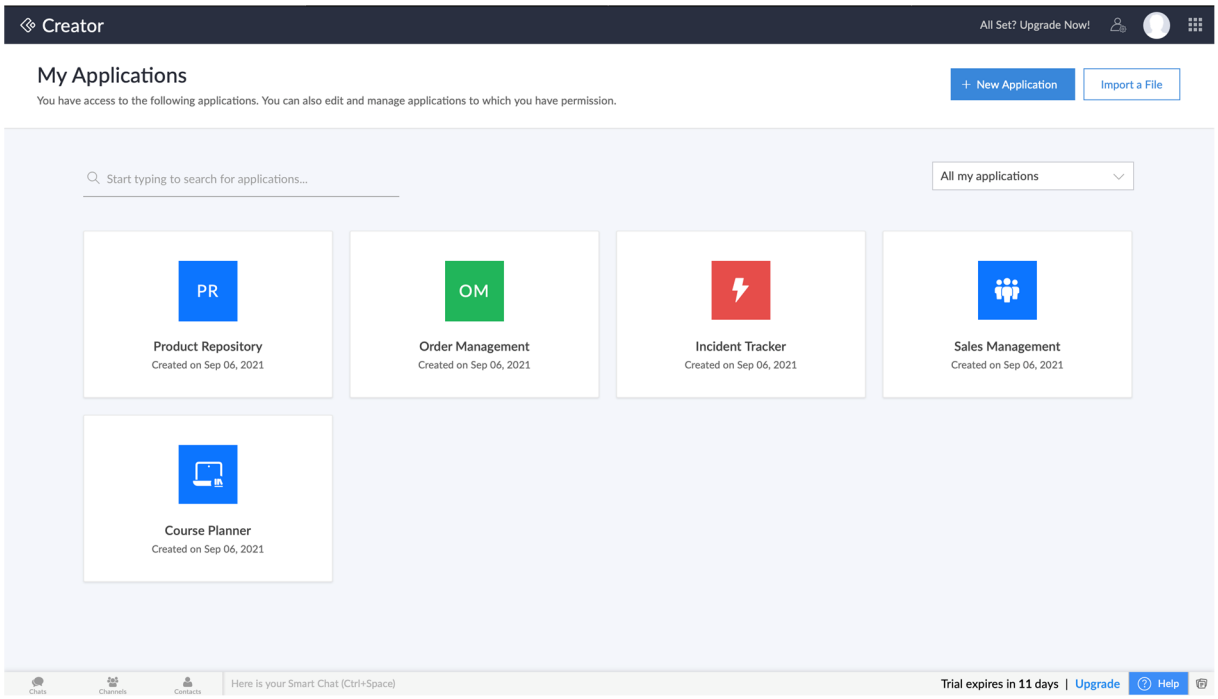

Figure 46: Zoho Creator home screen

Static Perspective. Entity types can only be created through GUI "forms". For this purpose, Zoho Creator includes a "form builder", which is a drag-and-drop editor for a set of pre-defined fields (see figure 47). Fields are used to specify the attributes of an entity type. The Zoho Creator platform includes generic data types like text and number or business-oriented data types like currency. Numerous templates to define forms are available within Zoho Creator (see figure 48). Although it is evident that data is persisted by the platform itself, the persistence mechanisms are inaccessible. A "form" can also be auto-generated based on some spreadsheet import or a connection to a Salesforce, QuickBooks, or any other Zoho product domain. "Forms" can be associated with one another through lookups. Lookups can also be defined across applications. A visual representation of entity types and their interrelations can be accessed in the developer tools (see figure 49). The proprietary data modeling language can only display entity types as node and relations as edges. No additional tools to edit or work with the visual data model is offered, the definition of "forms" is the only option to configure entity types.

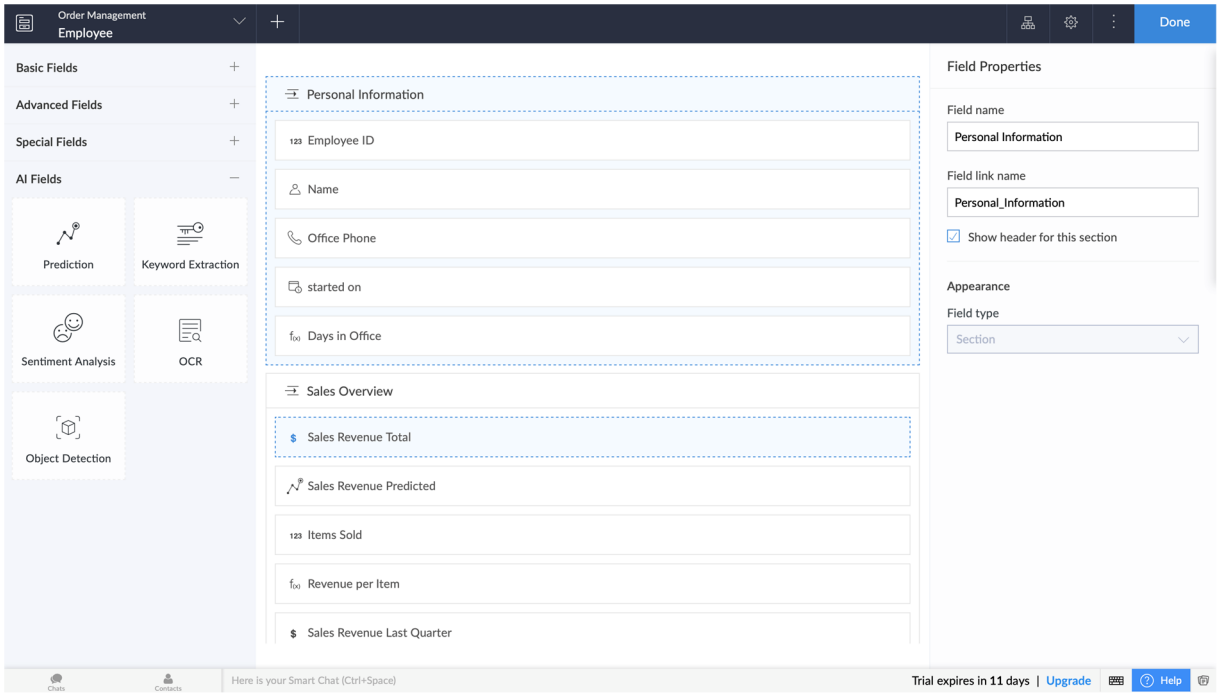

Figure 47: Zoho Creator form builder

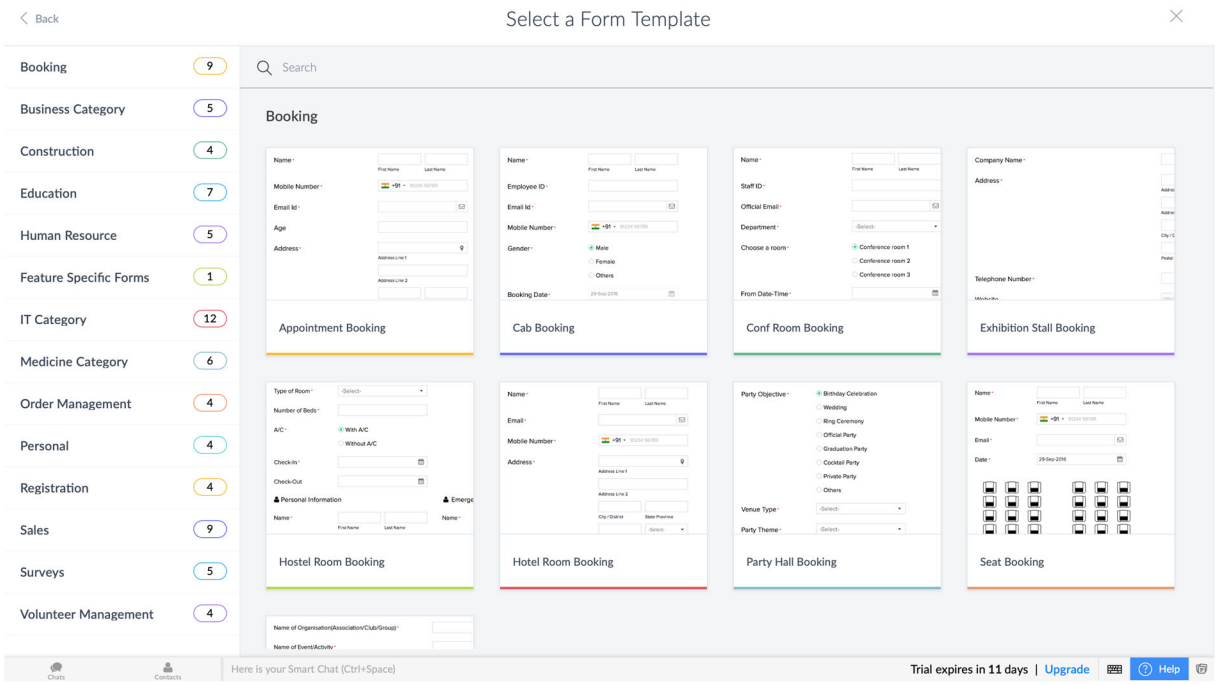

Figure 48: Zoho Creator form templates

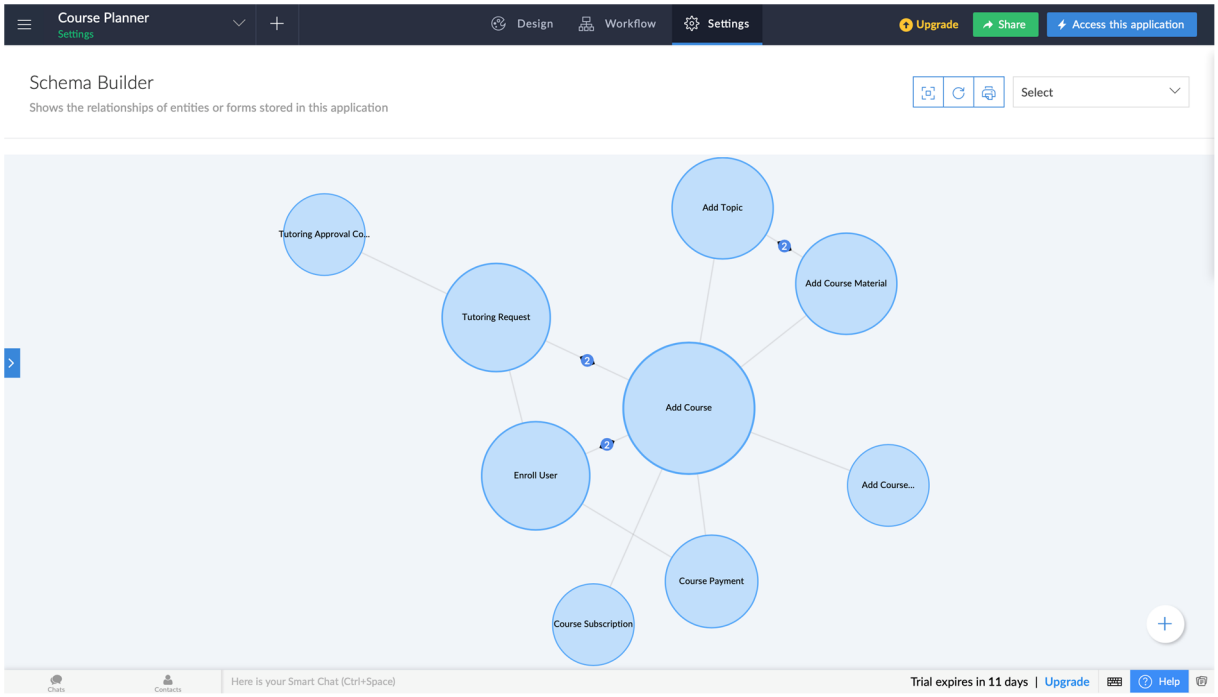

Figure 49: Zoho Creator schema builder

Table 45: Zoho Creator static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• GUI "forms" • data model	• proprietary	• diagram viewer • "forms" editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• fields can be specified according to generic data types (e.g., text, number), business-oriented data types (e.g., currency) • broad range of domain-specific GUI "forms"	• GUI "forms" can be interrelated through lookups	*implementation-level documents cannot be accessed*
Focus on Integration		
Common static abstractions across a range of applications	GUI "form" templates can support the specification of common static abstractions, but the choice of fields and entity types still belongs to the developer	
Access to common data repositories across a range of applications	It appears that all data is per default persisted through the Zoho platform	

Functional Perspective. Some basic CRUD-based functions are pre-implemented and can be used with the specification of different workflows (see *Dynamic Perspective*). Other than this, Zoho Creator enables the specification of custom function through built-in source code editors. Source code can be programmed in three languages: Java, JavaScript (Node.js), and Deluge – a proprietary programming language from Zoho. The auto-generated structure of a Java class is displayed in figure 50. Familiarity with Zoho APIs is required to access run-time objects. It is unclear how (or if) application-wide and cross-application variables can be defined. In the Deluge source code editor, some basic operations can be added via drag-and-drop (see figure 51; the list of operations is on the left).

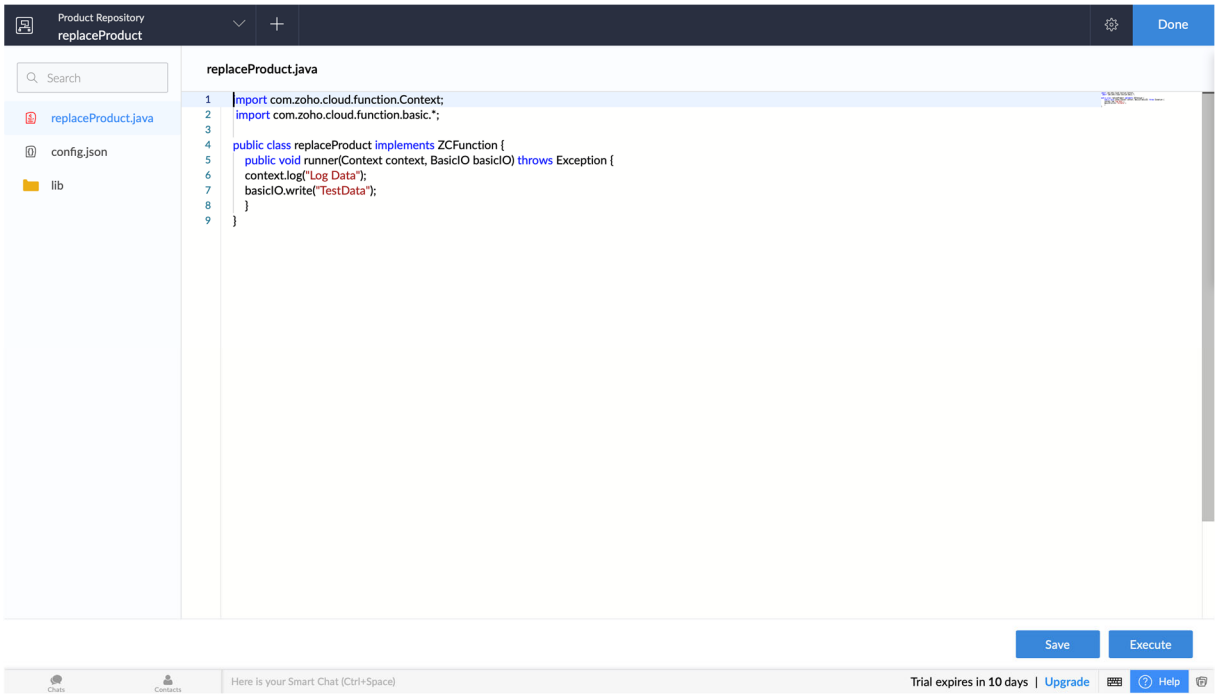

Figure 50: Zoho Creator auto-generated Java class

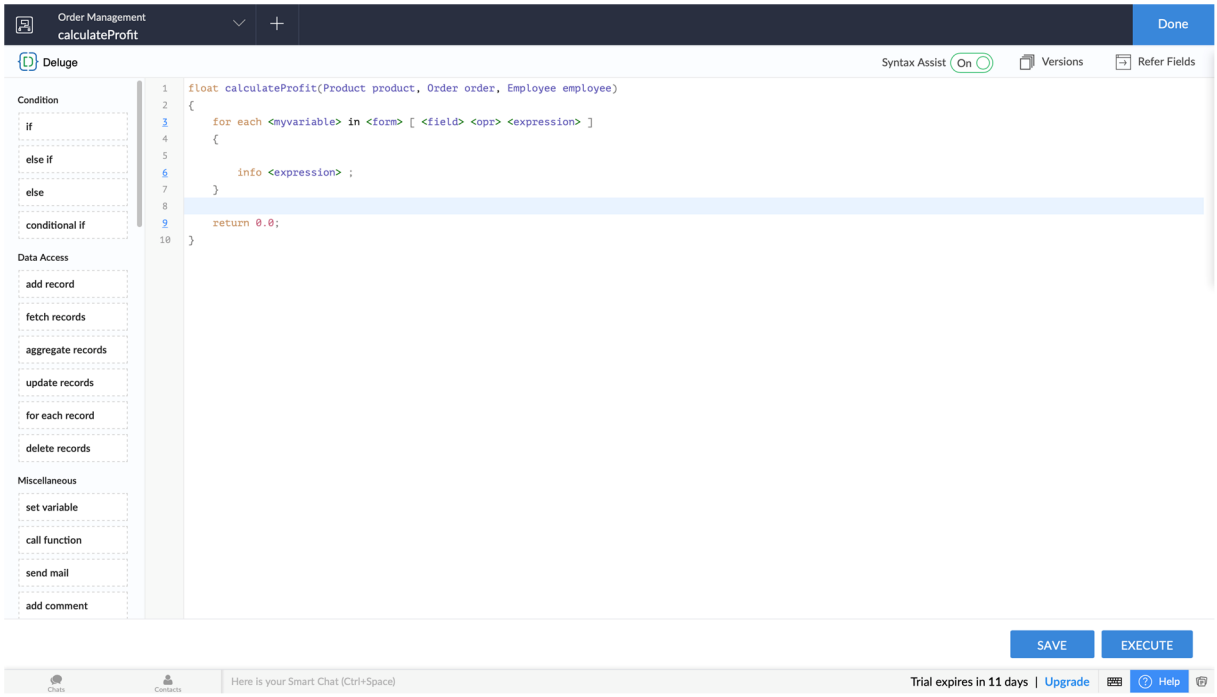

Figure 51: Zoho Creator Deluge script editor

Table 46: Zoho functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
no representation beyond source code	• source code can be edited in NodeJS, Java, and the proprietary Deluge language	• built-in source code editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• Deluge source code offers some basic operations	• composition	• Zoho runtime APIs • Java, NodeJS, Deluge source code files

Dynamic Perspective. In Zoho Creator, one of six different "workflows" can be implemented (see figure 52). "On a form event", "on a scheduled date", and "on approval" share an identical editor. The only difference between workflows of these categories is that they are triggered by different events. For example, "on a form event" must be triggered by some form-related CRUD event. Workflow types from these categories include sequential steps of "actions" and conditions (see figure 52). The definition of actions and conditions does not require any source code editing. The set of available actions depends on the "workflow" category selected by the developer. Six actions (e.g., disable field, hide field, define field value) can be defined for "on a form event" workflow types. The "on a function call" button (see figure 52) simply redirects to a source code editor that enables the specification of custom functions (see *Functional Perspective*). These functions are not inherently connected with any event. "On a business process" workflow types are managed in another editor (see figure 54). The proprietary modeling language only distinguishes between stages and transformations. Stages do not contain any further information other than their name. Transformations denote actions that are to be executed by a pre-defined set of users and automatic actions to be performed after the transition, e.g., a user needs to input some string which is thereafter transformed to lowercase. These workflows must be restricted to a single entity type. Reference models are merely available as part of the selected reference applications within Zoho Creator. The source code of a workflow type can be viewed in the proprietary Deluge programming language within the developer tools section of an application.

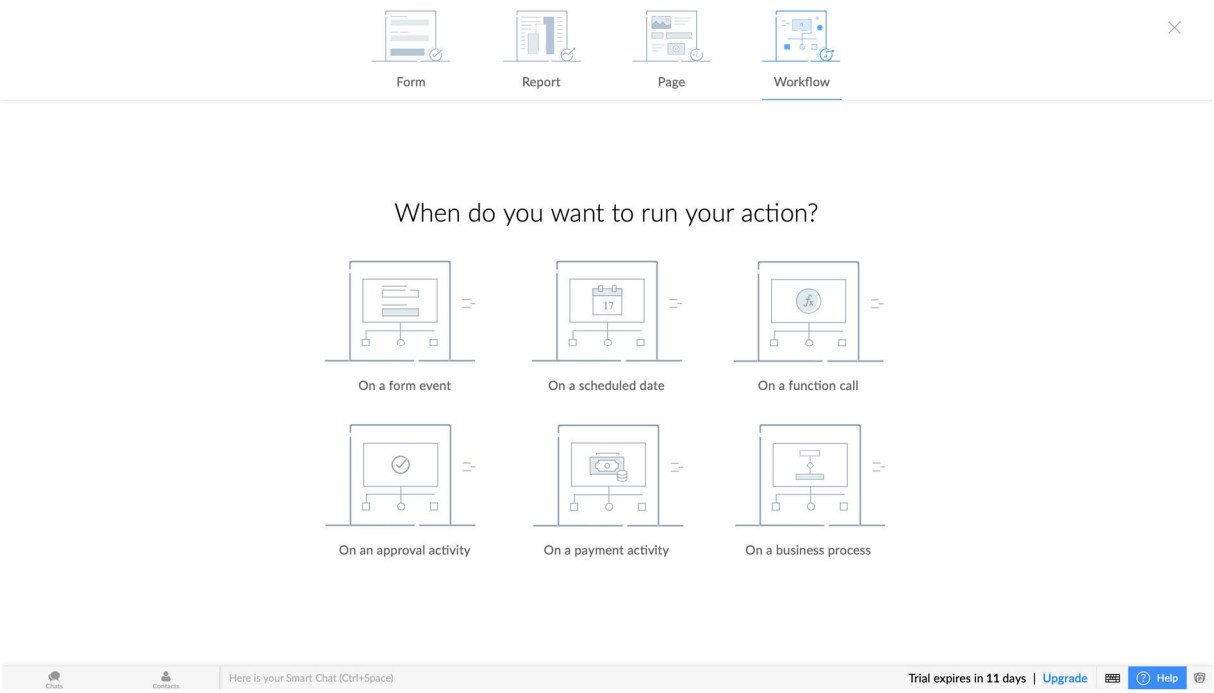

Figure 52: Zoho Creator workflow metatypes

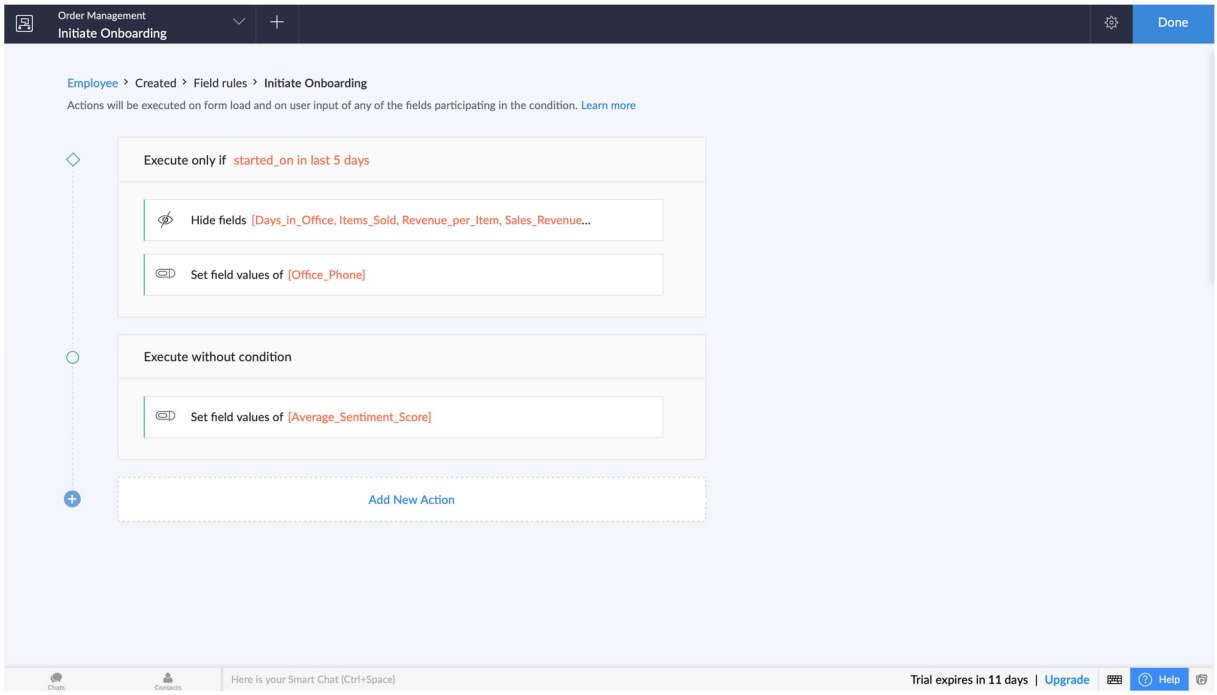

Figure 53: Zoho Creator "on a form event" workflow type example

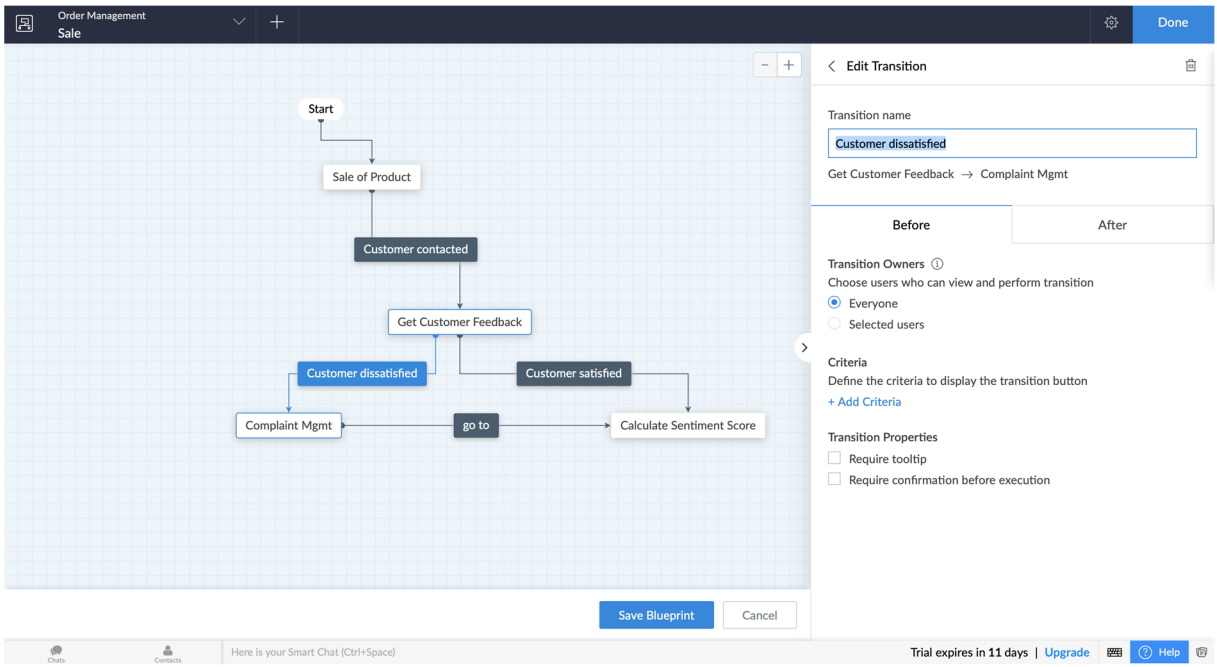

Figure 54: Zoho Creator "on a business process" workflow type example

Table 47: Zoho Creator dynamic perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• sequence of linear steps • generic process model	• proprietary	• diagram editor for "on a business process" workflow types
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• reference workflow types are available as part of reference applications	• generic concepts like stage, action, and condition • composition	• Deluge source code can be accessed

GUI Development. Three types of GUI pages are distinguished within Zoho Creator: "forms", "reports", and "pages". "Forms" are used to specify entity types as elaborated in the *Static Perspective* analysis category. The creation of a "form" yields a section with two auto-generated GUI pages (not to be confused with the "pages" of Zoho Creator): one to add a new entity and one that provides a tabular overview of all persisted entities. Additional GUI pages for "form" sections, i.e. a GUI page for the same entity type, can be defined through a Zoho Creator "page" (see figure 55). The third type of GUI pages, i.e., a "report", is not restricted to a single entity type and also not assigned to any of the auto-generated "form" sections. One of eight different report templates can be selected (e.g., pivot table, calendar, spreadsheet, and Kanban) The navigation panel can also be edited to rearrange sections and GUI pages. A drag-and-drop editor with a predefined set of GUI elements is provided for all three types of GUI pages. All GUI pages can also be accessed in the "developer tools" and edited in the proprietary ZML markup language (see figure 56). ZML provides native support for Deluge scripts. GUI screens are automatically adjusted to different screen sizes. Smartphone and tablet views can be edited separately.

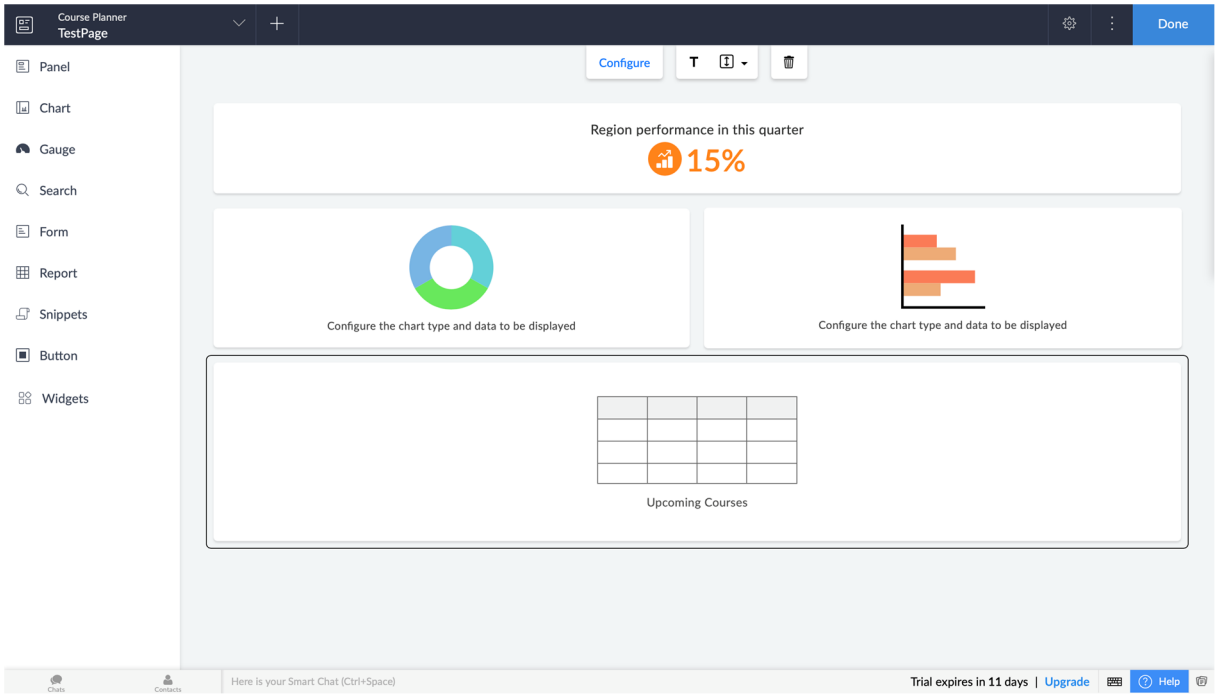

Figure 55: Zoho Creator GUI "page"

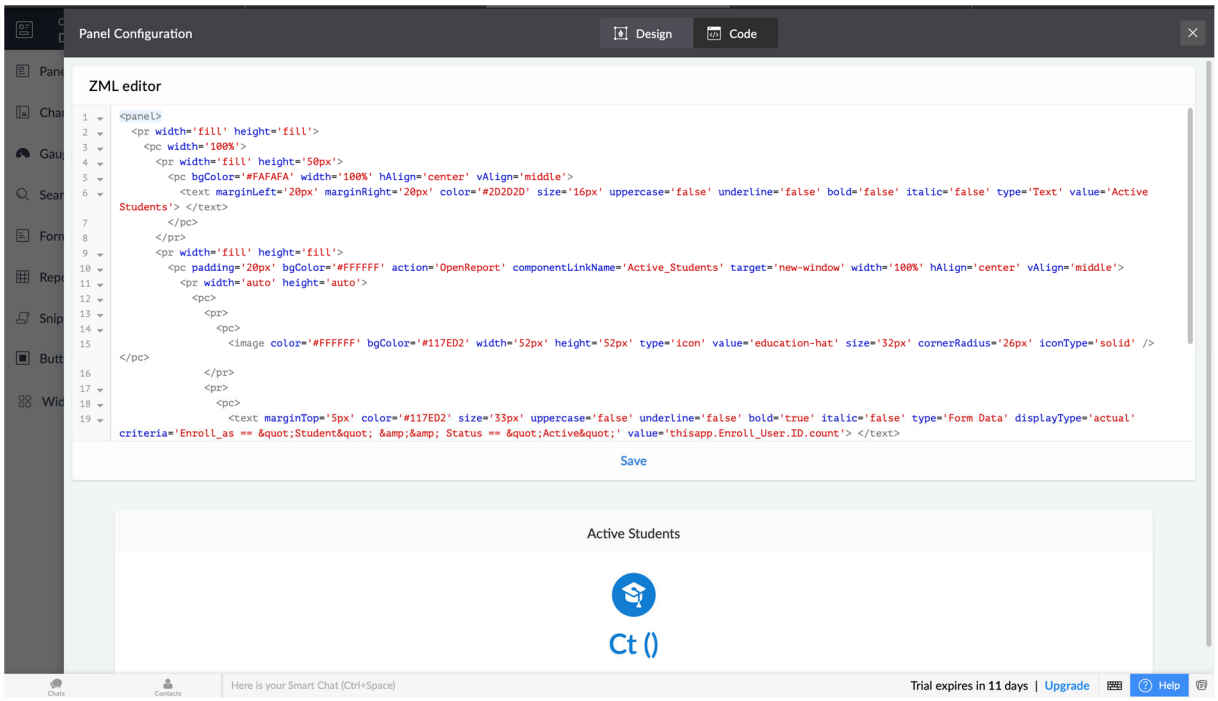

Figure 56: Zoho Creator ZML editor

Table 48: Zoho Creator GUI development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*
• GUI models	• drag-and-drop GUI Editor	GUI "forms" are a representation of entity types
Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• library of general-purpose GUI elements	• GUI elements and templates can be adjusted according to their particular style	

Further Aspects. Applications are deployed by Zoho and can be accessed via a common web server or through the Zoho Creator mobile application. It is also possible to define a separate domain for access by (external) users with restricted views on the application. Data persisted on the platform can be exported through known formats such as CSV. It is not possible to export entire applications.

Zoho Creator does not offer any explicit support for collaborative application development. Collaborative use of the platform is mainly addressed through the definition of roles and permissions. Permission types define CRUD-based access rights. Role types shall resemble organizational roles and can be arranged hierarchically. Role types are referenced in "on a business process" workflow types. A new user is assigned to exactly one role and permission type.

Zoho Creator offers some AI services as "AI fields" available in the "forms" GUI editor (all five available options are visible in figure 48). For each of the AI fields, the inner-workings are completely inaccessible. This includes the underlying ML model (AI seems to be understood synonymously to inductive ML algorithms), the data population used for training, and the estimated accuracy of a prediction. Figure 57 shows an overview of data entries where the AI field values are embedded ("Sentiment" and "Object Detection"). The values are hardly convincing. Prediction fields require a definition of one dependent ("target fields") and multiple independent ("predictor fields") dimensions. Apart from this, the calculation process cannot be accessed – it is even excluded from the respective Deluge script. The integration of external AI services is not specifically supported.

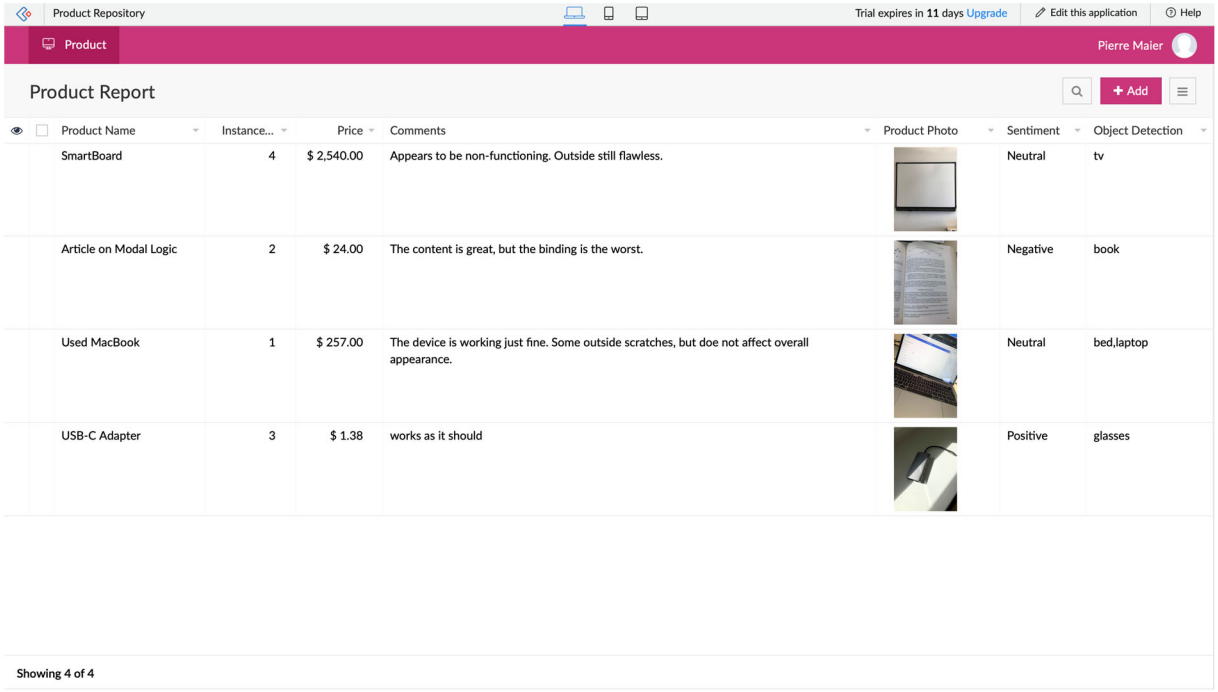

Figure 57: Zoho Creator AI fields

Table 49: Zoho Creator further aspects summary

Accessibility and Convenience of Use	
• no methodical application development support • fairly convenient and easy-to-use • access to source code in proprietary programming language for more advanced users • use of largely inaccessible, proprietary modeling languages	
Modes of Use	
Platform (i.e., Application Development)	*Application*
• no further support for collaborative application development	• distinction of role and permission types • role types shall resemble organizational roles and are used in some workflow types • permission types specify CRUD rights
Focus on Deployment and Scalability	
Target Location	*Support*
• cloud-based platform, accessible via a regular web browser or Zoho app for mobile devices	*no further accessible support mechanisms*
Focus on Artificial Intelligence	
• pre-implemented, inaccessible AI “fields” in GUI “forms” that produce ML classifications not convincing	

5.3.3.3 Zoho Creator: Conclusion

Emphasized Areas of Application Development. Most features of the Zoho Creator platform address the development of different kinds of GUI pages and workflow types. Data modeling features are only rudimentary, with no apparent support for integrating external sources or access to any sophisticated visual representation. Support for collaborative development or use is limited. It is only possible to define organizational roles and corresponding CRUD-based roles. Custom functions can be specified in different programming languages. For this purpose, a proprietary markup language (ZML) and programming language (Deluge) is provided. AI functionalities are also embedded through pre-implemented fields for entity types. However, the calculated values, alongside the inaccessibility of the AI model's functionality, make the incorporated AI features not convincing.

Provision of Abstractions. Zoho Creator fades out implementation-related details. Although the source code for most generated artefacts can be viewed and edited, some implementation-related aspects are still inaccessible. An example for this is the internal persistence of data and the functionality of AI services. Furthermore, numerous domain-specific abstractions in the form of GUI templates are offered within Zoho Creator. GUI pages are also the only mechanism to define entity types.

Role of IT Professionals. The LCP can be used by lay and professional developers alike. Superficial use with regards to the definition of entity types, workflow models, and GUI pages does not require in-depth knowledge of application development. However, this applies only in cases where the system-defined functionalities are sufficient. Professional developers would be required to implement additional operations via Java, Node.js, or the proprietary Deluge language. More advanced concepts associated, e.g., with integration concerns (inclusion of external data sources or API access) are generally not supported by the platform. It seems that such features demand additional Zoho products.

5.3.4 "Low-Code" Extended, GUI-, and Data-centric IDEs: Conclusion

The three considered LCPs of this prototypical category comprise a broad variety of different features and focal points. Mendix Studio Pro is a locally deployed IDE, WaveMaker omits any kind of workflow modeling, and Zoho Creator relies on a proprietary programming language. At the same time, a few commonalities can be identified which demarcate the LCPs from the platform of the preceding categories.

All three LCPs promote the use of source code. WaveMaker and Zoho Creator include several built-in source code editors. Mendix Studio Pro provides mechanisms to integrate external source code files within the Mendix environment. The high prominence of source code broadens the scope of feasible applications, but also demands developers to familiarize with the available programming languages, runtime APIs, and platform architecture. In this regard, it

is noticeable that all of the three platform vendors market their platform to support professional developers. This marks a conspicuous difference to the LCPs considered previously. While WaveMaker addresses only professional developers in their marketing statements, Zoho and Mendix promote productivity increase for professional and lay developers ("citizen developers") alike. Therefore, these platforms need to support collaboration of different kinds of developers. Mendix addresses this requirement explicitly through the provision of two development environments with different sets of features. User stories and comments, that are available in both environments, serve to support collaboration between the two kinds of developers. In Zoho Creator, a separate "developer tools" section enables access to more advanced development options. The separation is not as strict as for Mendix, though, and source code configuration is also part of other features of the platform, e.g., its GUI editor.

To achieve productivity increase, the three platforms provide additional representations beyond source code to develop and configure applications. Conceptual models of particular relevance in this regard. Mendix Studio Pro relies almost exclusively on the specification of different kinds of data and workflow models to develop applications. Zoho Creator offers different workflow modeling languages, although its proprietary modeling language for static abstractions is rudimentary. Similarly, domain-specific abstractions are addressed to some degree in Mendix and Zoho Creator. Mendix provides a marketplace where domain-specific add-ons can be acquired. Zoho Creator includes a vast catalog of reference templates for entity types and GUI pages.

5.4 Multi-Use Platforms for Business Application Configuration, Integration, and Development

The final prototypical category of LCPs includes comprehensive platform solutions that cover a broad range of different features. The characteristics relevant for the proposed category allocation is (i) application lifecycle support beyond one-time development and use and (ii) integration of numerous internal and external service to increase development productivity.

5.4.1 Microsoft Power Apps

Since Microsoft is presumably well-known, there is no need for even a short description of the company.

5.4.1.1 Microsoft Power Apps: Profile of Vendor

Product Portfolio. Microsoft offers numerous different software products for a broad range of different purposes. Among others, this includes the so-called *Power Platform* which includes four products: Power BI, Power Apps, Power Automate, and Power Virtual Agents. Microsoft Power Apps is Microsoft's LCP to build and manage applications. A specific characteristic of Power Apps is the close integration with other Microsoft products.

With respect to the focus of our investigation, it is worth noting that Microsoft's product portfolio represents a remarkable asset. It comprises a comprehensive stack of software systems including operating systems, database systems, ERP systems, a wide range of office applications, and various software development environments. Many functions provided by these systems can be reused to promote software development productivity.

Product Provenance. Microsoft has offered software development environments (SDE) for long. The specific strength of these systems was their integration with the Microsoft software ecosystem. In addition, the SDEs cover various programming languages, the integration of which is supported by a common language specification. While Microsoft is the leading vendor of office application systems, it has not managed so far to achieve a comparable share of the market for business applications. It seems that one cornerstone of Microsoft's strategy for increasing its base of business customers is to focus on support for software development and convenient integration with other products, including those from competitors. According to the historical presence of Microsoft Power Apps on the web, the platform is advertised at least since early 2016 (Jan 09, 2016). While the label "low-code" was not used at first, the product was already promising features which appear characteristic for current LCPs, such as convenient integration with common databases and enterprise systems.

Focus on Marketing The selling proposition of Power Apps is focusing on four main features: (1) Contribution to organizational agility by "rapidly building low-code apps that modernize processes and solve tough challenges".[57] Related to the previous: (2) promoting development productivity by "launching apps right away using prebuilt templates, drag-and-drop simplicity, and quick deployment".[58] (3) Convenient integration with other systems through "connectors". (4) Support of lay developers: "Empower everyone to build apps. Building apps with Power Apps helps everyone from business analysts to professional developers work more efficiently together."[59] In this context, Microsoft also uses the term "citizen developer".[60] Further features that are regularly mentioned comprise portability across different platforms such as PCs, tablets, and smartphones, deployment in the cloud, and "prebuild AI components".

The messages sent by Microsoft marketing give the impression that it is a tool which can be used for the quick development of any kind of (business) application, covering a wide range

[57] https://powerapps.microsoft.com/en-us/, accessed 09-08-2021

[58] https://powerapps.microsoft.com/en-us/, accessed 09-08-2021

[59] https://powerapps.microsoft.com/en-us/build-powerapps/, accessed 09-08-2021

[60] https://www.microsoft.com/en-us/insidetrack/citizen-developers-use-microsoft-power-apps-to-build-an-intelligent-launch-assistant, accessed 09-08-2021

from small solutions for specific use cases to distributed, process-driven enterprise-level systems; a tool that is highly effective for professional developers and lay developers as well. It is also suggested that it is the tool of choice for integrating apps with existing systems. However, the picture of a universal, "one size fits all" software development environment is misleading. Otherwise, there would be no need for other SDEs in Microsoft's portfolio, such as the "Visual Studio" product line. Therefore, the only way to develop a better understanding of purposes, Power Apps is especially suited to server, is a closer look at its features.

Table 50: Microsoft Power Apps profile of vendor summary

Product Portfolio
• broad range of development tools • LCP as part of "Power" suite of business application development support tools
Product Provenance
• platform emerged in 2016 • no specific developments can be noted from its archived web presence
Focus on Marketing
• emphasis on increased speed of development, support for lay developers, and integration with external systems • target lay and professional developers and improved collaboration between them

5.4.1.2 Microsoft Power Apps: Analysis of Platform Features

The development environment ("PowerApps Studio") runs in a web browser. The home screen presents two principal approaches to developing an app. One can start with data ("Start from data"), which means to select an existing data source. In contrast, the second principal approach "Make your own app" comprises three specific approaches. One may start with the design of a GUI ("canvas app from blank"), the design of models ("model-driven app from blank"), or "create a website to share data with external and internal users" ("portal from blank"). Note that we did not check "portal from blank", because it was not included in the test version. The home screen allows to choose from the principal development modes, gives an overview of existing projects and access to tutorials for different use cases and experience levels (see figure 58).

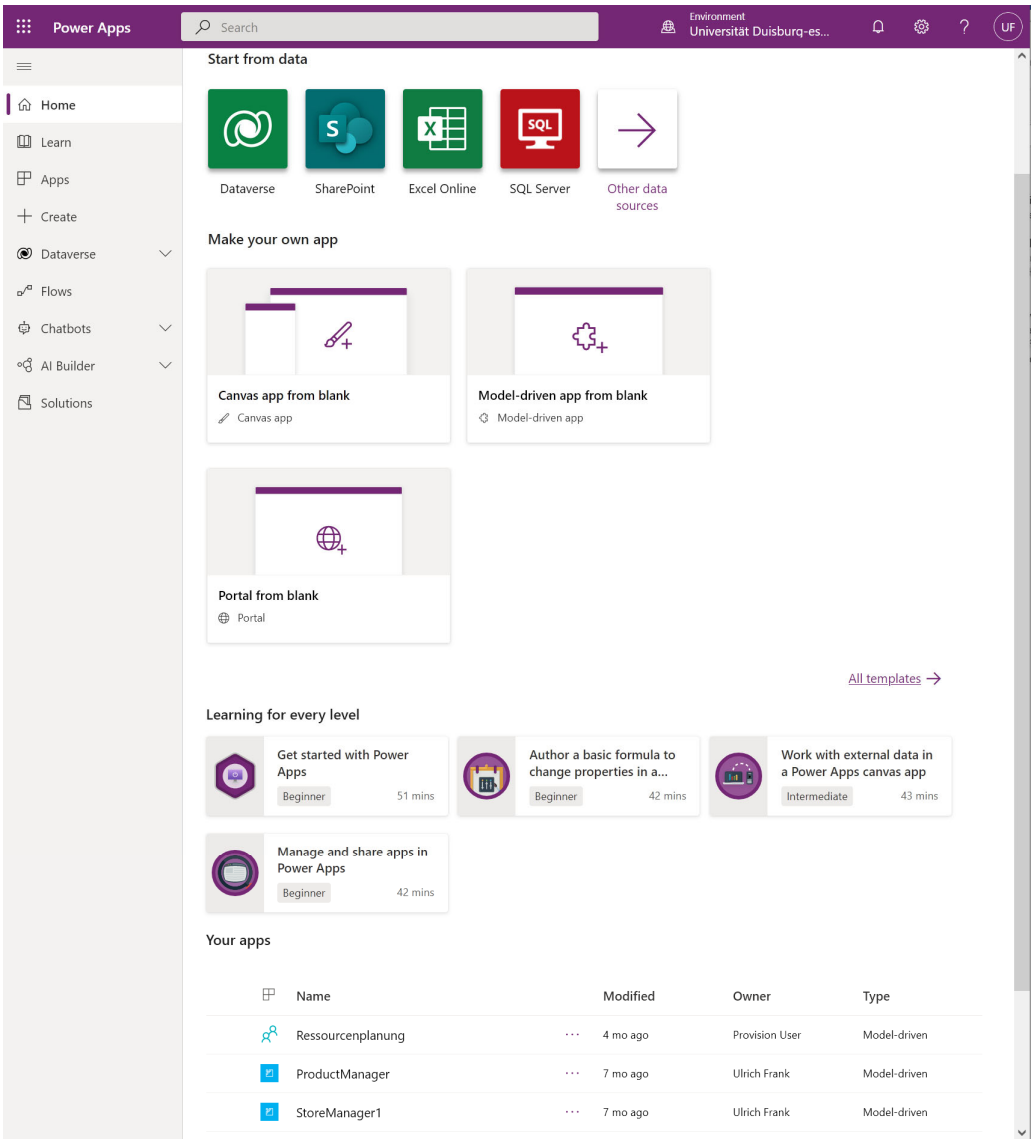

Figure 58: Microsoft Power Apps home screen

Static Perspective. Data are of outstanding importance for using Power Apps. The "start from data" mode requires existing data sources. Data sources may be provided by services offered by Microsoft. They comprise *SharePoint*, *Excel Online*, *SQL server* and *Dataverse*, formerly offered as *Common Data Service*. Dataverse allows accessing data from various application systems, including the Microsoft enterprise software *Dynamics 365*. In addition, other data sources like Excel files stored on *DropBox* or *OneDrive* can be used, too.

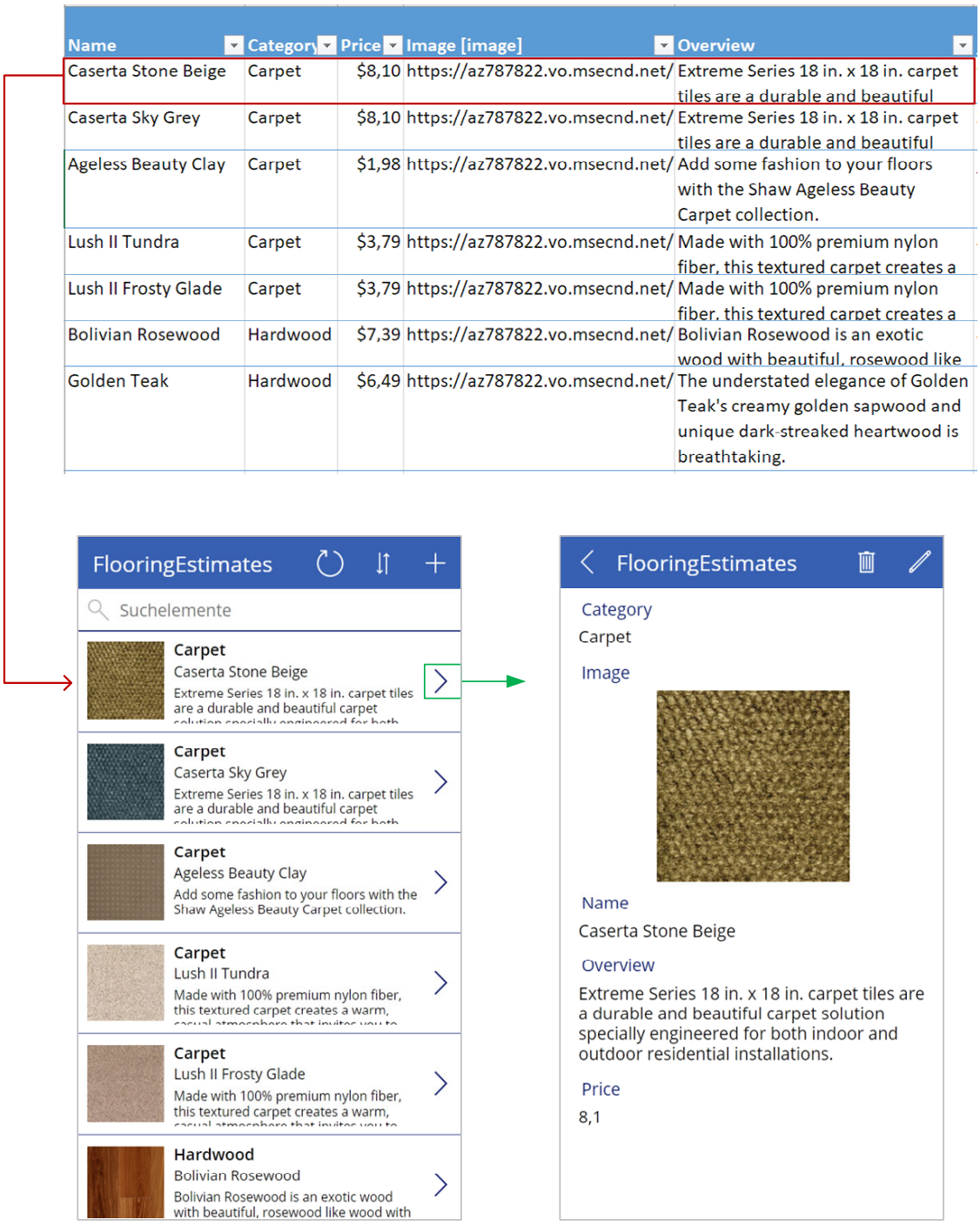

Figure 59: Microsoft Power Apps mapping of table to screens of app

"Start from data" is based on the idea to map a table to smartphone or tablet screens. The initial mapping is done automatically and includes already generic navigation capabilities. This allows for creating apps considerably fast. As soon as Power Apps is connected to the table, the app is executable. It consists of three screens. A "browse" screen shows a list of all items in the table. Every entry to the list corresponds to one row. A "detail screen" serves a more comprehensive representation of the elements of a tuple. Finally, an "edit screen" allows editing the

elements. The example shown in figure 59 illustrates the mapping from a table to a default "browse" screen and its connection to a default "detail" screen. The mapping includes some "magic". First, there are rules that define the layout of screens. They seem to define the placement and size of an image. However, the details of such a rule remain unclear. If, e.g., a second column of images is added to the table, it is displayed smaller. Second, there is (almost) no need to specify data types. That corresponds to Excel. Numbers are interpreted as such based on their representations as strings. Only images have to be defined as such by adding the keyword "Image" in brackets to the name of the corresponding column.

Such rapid success is not very impressive on closer inspection, however. This approach to creating an app is restricted to sources that include one table only. Furthermore, any formulas that might be defined with the table in Excel are lost. Hence, the functionality of an app generated from a table is limited and would be better supported by Excel. However, the generated app is a starting point only for further customization. That may include changing the layout of the GUI or adding functions.

The "canvas app from blank" entry starts with an empty canvas that can be filled with multiple widgets from the GUI builder (see *GUI Development*). On closer inspection it is very similar to "start with data", because it also requires connecting to an existing data source. If the mapping is done at the beginning, and the "catalogue" mode is selected, it results in an automatically created GUI that corresponds to that of the previous approach. The mode "form" requires a little more effort, but gives greater freedom of design. At first, one selects the elements of the columns from the previously connected table that should be displayed on the screen. Power Apps creates widgets and distributes them on the screen, where the layout can be rearranged at will.

The "model-driven" mode supports different approaches to designing an app. One can start with designing a GUI. Different from the previous approaches, these are not restricted to smartphones or tablets. Subsequently, data sources can be added. A more powerful approach is based on a predefined structure of application elements. It comprises data ("entity view") and, as an option, business processes ("process flow view"). The effectiveness of this approach depends on reusing existing apps or existing tables. The trial version that was at our disposal comprises various, in part rather specific apps and a considerable number of tables from a general business domain.

The term "model-driven" is used in a rather idiosyncratic way. First, there is hardly any graphical representations of models available – with the exception of process models. Second, the terminology suggested by Microsoft hardly fits the idea of conceptual modeling. Instead, it rather corresponds to database design, e.g., by referring to database keys. It also goes beyond data modeling by referring to technical aspects of the system to be developed. For example, to further specify the predefined entity type "Address", one may select features like "mail

merge" or "queue" (without being provided with an idea of what that is supposed to mean). This is surprising with respect to the claim of providing a tool for lay developers.

The model of an app is presented in the structure shown in figure 60. Note that changing the language to English did not apply to every screen. Every row represents an entity type, the name of which is shown on the left side of the row, printed in white on blue. In addition to the data view ("entity view"), forms can be created to define a user interface. Diagrams serve the graphical representation of data. Dashboards are used to provide collections of information at real-time. To "model" data, one adds an entity type ("Adresse" in the example).

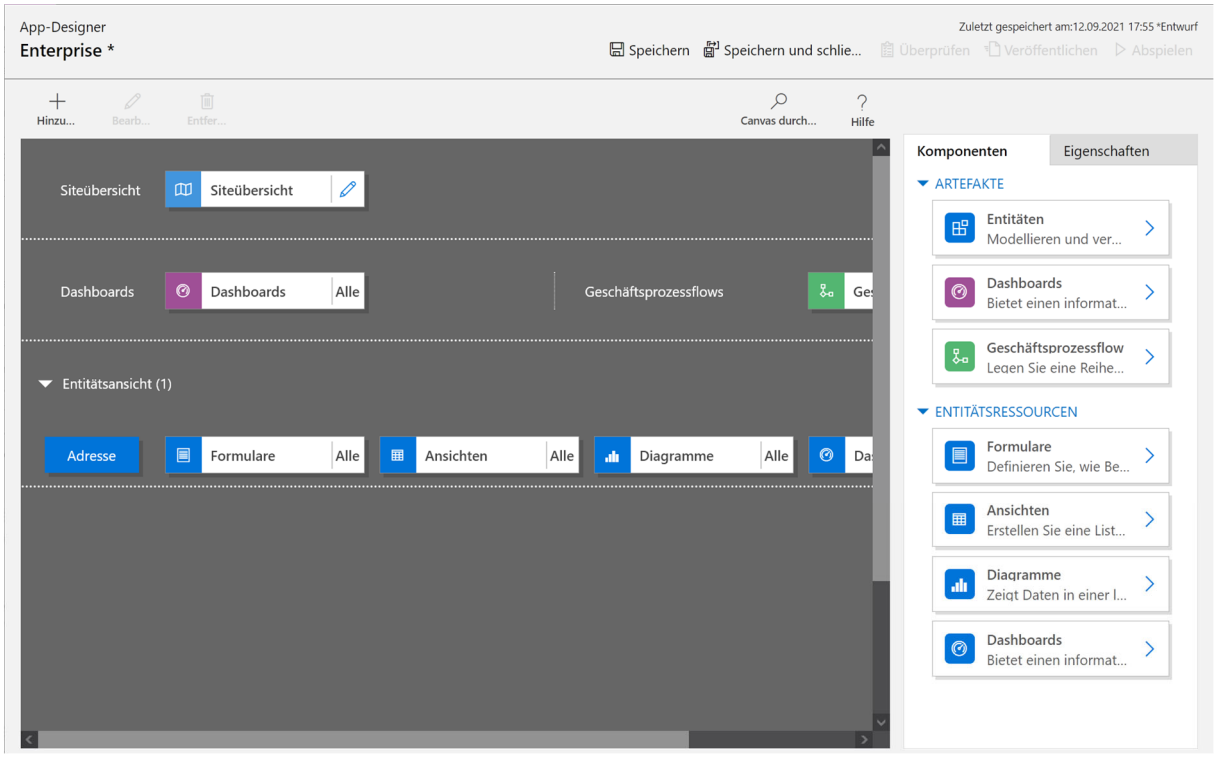

Figure 60: Microsoft Power Apps app designer in "model-driven" mode

Subsequently, an editor opens that allows to specify the properties of the entity type. To this end, the user is presented with a rather technical view that might be appropriate for a system administrator, but hardly for a novice user (see figure 61).

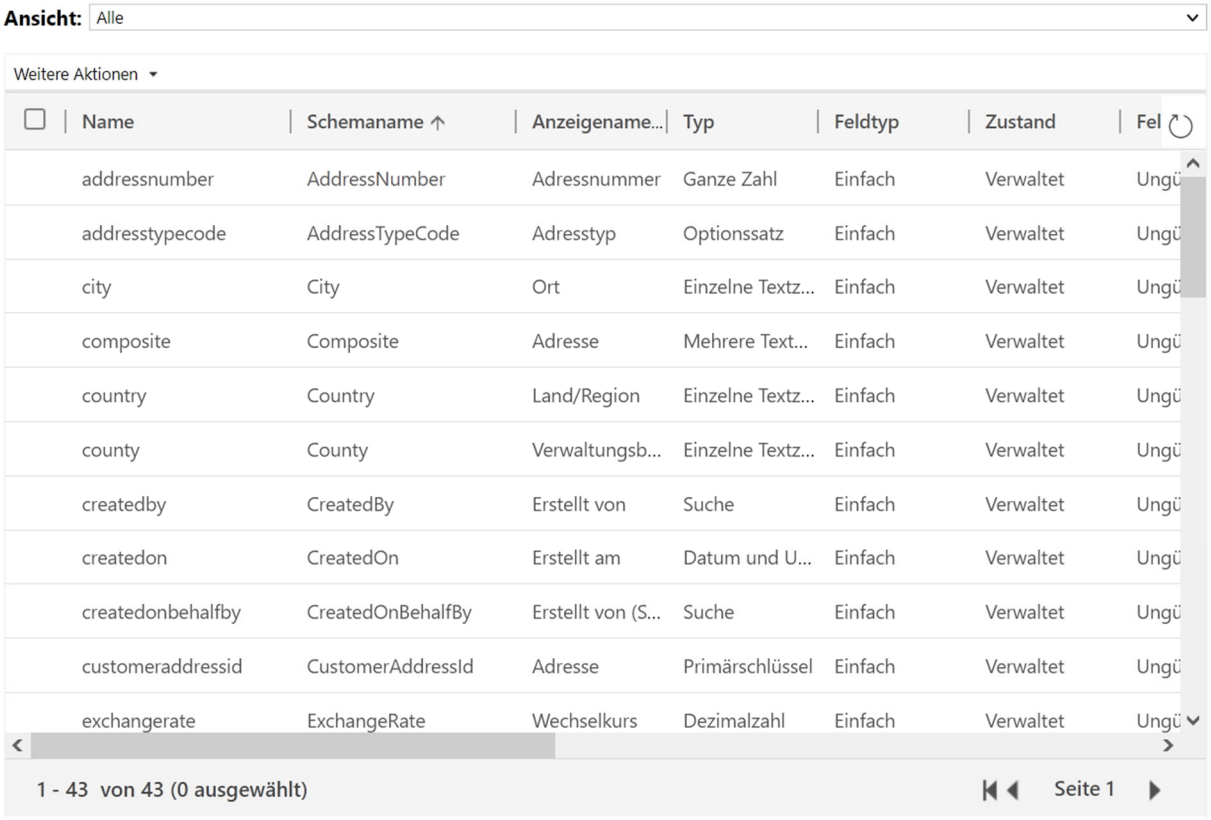

Figure 61: Microsoft Power Apps specification of entity type

Furthermore, views may be available that address specific use cases such as searching for elements. Predefined diagrams serve to graphically illustrate the available data. Finally, dashboards allow for the online visualization of selected data.

Table 51: Microsoft Power Apps static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• data types • data "model" represented by tables and GUIs	• Even though the platform offers a "data modeling" function, no data modeling language is included.	• data definition component
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic abstractions • domain-specific abstractions like "Address"	• entity types, tables	• DBMS • various other Microsoft products
Focus on Integration		
Common static abstractions across a range of applications	various applications may share abstractions defined with other Microsoft systems, e.g., a database schema.	
Access to common data repositories across a range of applications	supported by using data stores provided by other Microsoft software systems such as SharePoint, Excel Online, SQL server and Dataverse	

Functional Perspective. It seems that PowerApps does not directly allow coding. Instead, it provides reusable functions. First, there are functions for the implementation of GUIs, e.g., the navigation between the screens of an app. They are convenient to (re-) use, but hardly allow for modifications. Second, and much more powerful, are reusable artefacts that can be selected in the "model-driven" mode. They comprise components to create diagrams and dashboards. Furthermore, it is possible to integrate functions (or components) of other Microsoft systems such as the ERP system *Dynamics 365*. The reuse of these artefacts will usually imply reusing corresponding data structures, and, in order to avoid data redundancy, common data stores like Dataverse. While the reuse of complex artefacts is suited to boost the productivity of developing an app, that does not mean that this kind of composition is convenient. Instead, it is rather cumbersome and requires some knowledge of software architectures.

Table 52: Microsoft Power Apps functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• no specific representation	• no specific language • code	• GUI editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• multiple functions that can be added to GUIs • specification through code		• Functions provided by various Microsoft products can be reused.

Dynamic Perspective. The design and execution of business processes (or workflows respectively) is supported in the "model-driven" mode. It allows the design of business processes with a proprietary modeling language (see figure 63). In order to prepare a process model for execution, it is necessary to define the data and/or components required by each activity. Furthermore, one needs to specify conditions that control alternative threads of execution. Both seems to be possible but is far from intuitive. The GUI designer should allow for the specification of GUIs used within each activity of a process. But we did not find a corresponding feature either. This lack of "user experience", the importance of which Microsoft repeatedly emphasizes, may be contributed to the fact that the process designer is not an inherent part of Power Apps. Instead, it is part of Dynamics 365, which gives rise to the assumption that the effective use of the process designer requires knowledge of Dynamics 365.

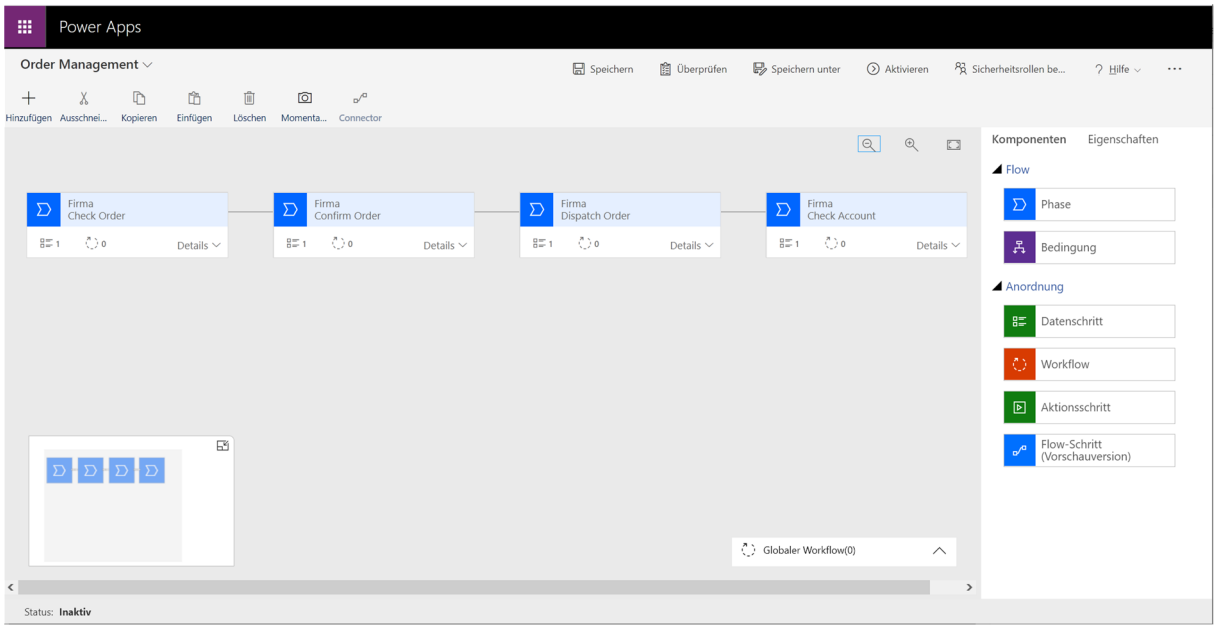

Figure 62: Microsoft Power Apps process designer

Table 53: Microsoft Power Apps dynamic perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• business process models	• proprietary process modeling language	• model editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• reference process models	• process, activity (“phase”), branching *Comment*: It seems that the language lacks a concept to describe events.	• code (various programming languages supported by Microsoft)

GUI Development. Support for the design of user interfaces seems to be of particular relevance. PowerApps includes various GUI designers. The “catalogue” mode (figure 59) is based on a default mapping of a table to three screens of a phone (or a tablet). The GUI can be modified, but it is restricted to a given size. The “forms” mode allows for a more versatile approach. It is intended for the definitions of GUIs that are displayed in web browsers (see figure 63). It com-

prises the common range of widgets such as text boxes, list boxes, buttons, etc. It is also possible to add diagrams (similar to Excel) and to include components from other applications (more precisely: the visualization implemented with these components).

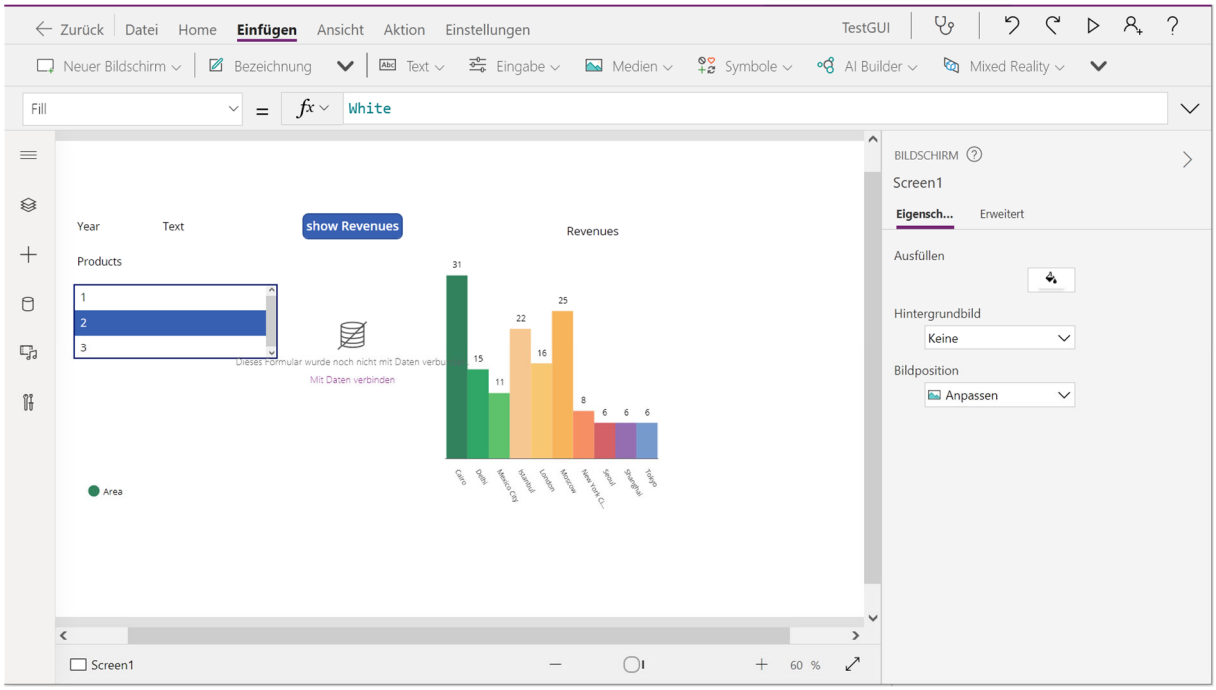

Figure 63: Microsoft Power Apps "Forms" mode GUI designer

The GUI designers for both, the "forms" and the "catalogue" mode, feature an execution mode. As soon as the GUI is connected to a data source, the app can be executed. This happens very conveniently by pressing a button or a key.

A further GUI builder is provided with the "model-driven" approach (see figure 65). It is based on the entity view and is intended to define forms for accessing fields of a table. It has fewer features than the "forms" GUI designer.

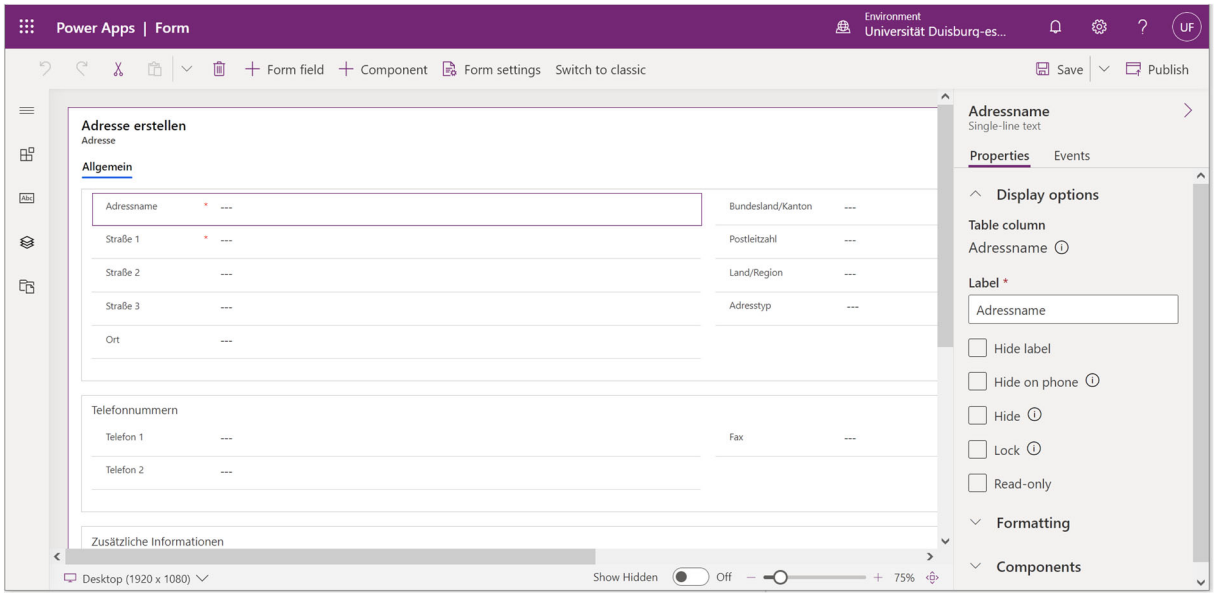

Figure 64: Microsoft Power Apps GUI designer in model-based approach

Table 54: Microsoft Power Apps GUI development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*

• GUI models
• generic GUI frameworks based on screens and forms
	• GUI Editor
	  ○ drag-and-drop
	  ○ WYSIWYG
	  ○ platform-independent style
		The architecture is widely hidden. It seems to implement the MVC pattern.

Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• library of general-purpose GUI elements • example applications with reusable GUIs	• customization of reference GUIs	

Further Aspects. Power Apps supports two principal modes of producing and deploying apps. An app can be created for running in a web browser, hence, on a wide range of platforms. The main platforms Power Apps targets are smartphones or tablets. To provide specific support for these platforms, runtime environments are available for Windows, iOS, and Android. As soon as an app created for this runtime environment is deployed to the cloud, it can be accessed and executed by mobile devices where the runtime environment is installed. Access is limited to authorized users, e.g., members of a certain project or employees of a company or organizational unit. In any case, an app and all the resources required to run it, are stored in a

Microsoft cloud. There is no indication that apps that are developed with Power Apps can be maintained with other SDEs, or executed in other runtime environments (with the exception of those that run in web browsers). There seems to be no version control. Collaborative editing of documents is apparently not supported. During run time, the potential for collaboration depends chiefly on the data sources used for the implementation of an app.

Microsoft Power Apps includes an "AI Builder" module to create and configure several ML models. An overview is provided in figure 65. As can be seen, the platform currently offers eleven pretrained ML models which can be used in applications and flows. The use of these models does not seem to demand any particular knowledge of ML algorithms. The model (hyper-)parameters are inaccessible as well as the underlying data used for training. Although this omits any decisive assessment of the model's capabilities, it is possible to test model predictions based on custom inputs with the respective degree of confidence calculated. The definition of ML models is thereby guided by a specific list of "best practices" that shall support reflecting upon reasonable use cases. For example, the "Text Recognition" pretrained ML model states that only images should be used with one column of text. The custom specification of ML models also avoids source code editing. Rather, one of five use-case specific ML models can be refined (the underlying ML model details are inaccessible) and stepwise configured in a user-friendly way (see figure 66). The impact of these configurations on the performance of an ML model remain unclear.

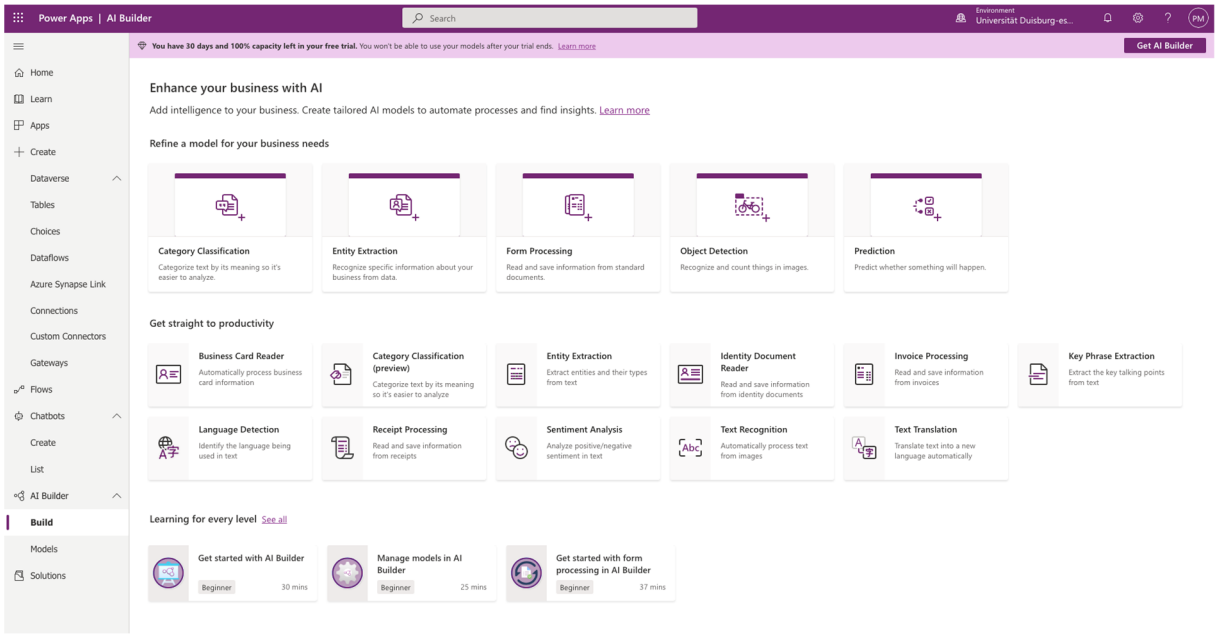

Figure 65: Microsoft Power Apps AI Builder

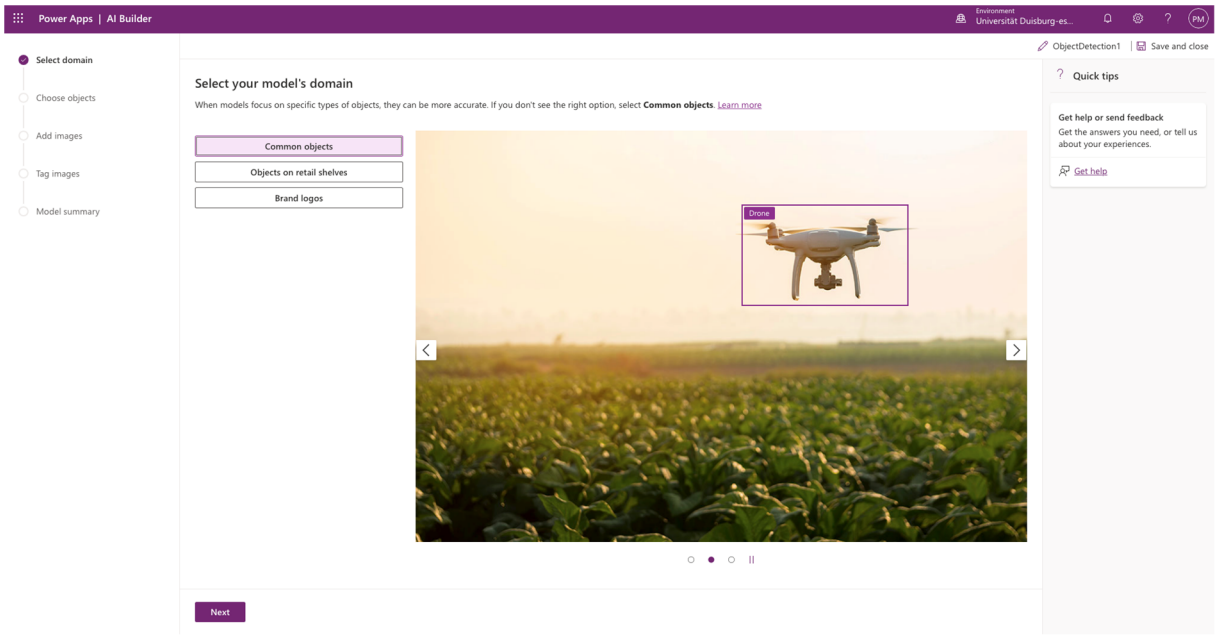

Figure 66: Microsoft Power Apps object detection model builder

Table 55: Microsoft Power Apps further aspects summary

Accessibility and Convenience of Use	

• guidance
  ○ no specific development method
  ○ modest guidance through different modes of use
• LCP user interface
  ○ based on proprietary widgets and general GUI framework
• offered modeling languages
  ○ proprietary process modeling language	

Modes of Use	
Platform (i.e., Application Development)	*Application*
• definition and management of access rights • conjoint use with other Microsoft IDEs	• definition and management of access rights • use in distributed environments • use in heterogeneous environments possible by running Power Apps on different platforms for which it is available.
Focus on Deployment and Scalability	
Target Location	*Support*
• development environment runs in web browser	• runtime systems available for Windows, Android and iOS
Focus on Artificial Intelligence	
• various pre-trained ML solutions, such as text recognition, are available with the platform	

5.4.1.3 Microsoft Power Apps: Conclusion

Emphasized Areas of Application Development. The feasible use cases for Microsoft Power Apps reflect the dual perspective on different kind of developers. On the one hand, it seems to focus on occasional developers with a very restrictive scope of adjustments, on the other hand, it tries to support the development of large application systems, which require not only substantial knowledge of software architecture, but also recommend programming skills. It is therefore unclear to us how Power Apps relates to other, professional SDEs offered by Microsoft. Nonetheless, in contrasts to other LCPs analyzed, Microsoft offers several mechanisms to support reusability of application artefacts and, to some degree, also support the integration with external data sources. This can contribute to a higher degree of integration, although this largely depends on the developer skills. Key features like data diagrams are missing and the offered domain-specific abstractions are hardly sufficient for comprehensive application development. Power Apps can thus be used to develop extensive application solutions with AI capabilities. The platform puts a clear emphasis on GUI configuration. For this purpose, three supplementing types of GUI editors are offered, serving different needs for various developers. Additionally, data and static features generally mark a core of Microsoft Power Apps. Although visual data modeling capabilities are not included, Power Apps tries to increase productivity by providing numerous reference entity types as well as support for various external data sources. While the tight integration with other Microsoft products creates a clear advantage with respect to productivity, it is likely to increase the dependency from the Microsoft ecosystem and, hence, to compromise protection of investment. PowerApps provides a few features to enable "quick wins" even for laypersons, like the automatic creation of simple apps from tables together with convenient ways to modify these apps. The exhaustive internal and user-friendly features to specify and configure ML models should also be highlighted.

Provision of Abstractions. Implementation-related details are largely faded out for the developer. Source code cannot be accessed nor edited. The provided abstractions, e.g., to define entity types are hardly sufficient for lay persons either. Noteworthy here is Microsoft's explicit use of the term "model-driven development", which is rather confusing when compared to the remarks from chapter 2. The approach followed by Microsoft Power Apps appears to follow the idea to rely on prebuilt application components. For this purpose, several domain-specific abstractions are offered. Entity types can be defined across Microsoft products and are shared with Microsoft's Dataverse, several are already provided by Microsoft. Additionally, numerous templates and example applications shall serve as a reference point for further application development. While the LCP does offer a variety of reusable artefacts of this kind, it is not too impressive and hardly sufficient for the development of many applications. However, it is not clear to us whether this is a restriction that applies to the test version only.

Role of IT professionals. Certain quick and simple applications can conveniently be developed by laypersons. This is exemplified in the "start from data" applications. This, however, marks

only a rather restricted scope of applications not feasible for most use cases: it relies on a single, independent entity type that can follow a predefined GUI structure. The role of professional developers is then to enhance these simple solutions to realize more extensive business needs and integrate the application with the system landscape. Effective use of the platform's features generally appears to demand professional developers. This contradicts the marketing statements from Microsoft itself, where extensive use by "citizen developers" is advertised.

5.4.2 Appian

Appian, the vendor of the platform with the same name, is a North American software company that has its headquarters in Virginia. It has about twenty offices in all continents except for Africa.

5.4.2.1 Appian: Profile of Vendor

Product Portfolio. The low-code platform is the only product of Appian in the company's product portfolio.

Product Provenance. The earliest archived web entry from the software company Appian (and not Appian Graphics which occupied the domain appian.com previously) dates back to Oct 25, 2005. Appian's platform is denoted as *Appian Enterprise Business Process Management Suite* which "simplifies the development, execution, and maintenance of business processes" (Oct 26, 2005).[61] Later marketing messages put emphasis on enterprise integration with the development and deployment of "composite applications" (Nov 02, 2006).[62] The Appian platform is to increase ease-of-use (Feb 27, 2008) and development speed (Jan 28, 2010). On Apr 10, 2011, support for the development of "dynamic and interactive web forms through a drag-and-drop interface" is advertised. Later, marketing of the Appian platform is centered around the notion of "worksocial", which is used to combine a "no coding process design" and "simple no-training interaction" (Aug 04, 2012). This shall support joint development of business and IT (Feb 27, 2018).[63] The platform is relabeled as a low-code platform at least since Jul 06, 2017 with no apparent change in the offered features.

Focus on Marketing. Appian's marketing messages focus on four promises: increased development speed, improved data integration capabilities, multi-experience development (shall be similar to multi-channel development only that "user experience" can be adjusted to different user channels), and simplified scalability.[64] Robotic process automation (RPA), BPM, case

[61] http://www.appian.com/AppianEnt/AE/appianEnt.html (archived)

[62] http://www.appian.com/AppianEnt/AE/appianEnt.html (archived)

[63] http://www.appian.com/product.jsp (archived)

[64] https://appian.com/platform/low-code-development/low-code-application-development.html, accessed 09-07-2021

management, and AI are also advertised as core components of the platform.[65] Appian does not specifically address a particular kind of developer. It does, however, suggest four prototypical roles for developers using their platform[66] – neither of which corresponds to lay developers.

Table 56: Appian profile of vendor summary

Product Portfolio
• LCP as the only product of Appian
Product Provenance
• early focus on BPM and quick development of "composite applications" • everyone should be enabled to develop applications, emphasis on collaboration between different organizational departments • relabeled as LCP since 2017
Focus on Marketing
• low-code development in Appian shall serve to support data integration and multi-experience development, among others • RPA and AI advertised as central features • no exclusive focus on any kind of developer

5.4.2.2 Appian: Analysis of Platform Features

The home screen of the *Appian Designer* is displayed in figure 67. The *Appian Designer* is the main module of the Appian platform and is also the focal point of the subsequent analysis. The other modules (*Cloud Database*, *Appian RPA*, *Quick Apps Designer*, and *Tempo*) provide some additional features. Each application consists of a customizable set of "design objects". Figure 68 presents an overview of design objects for an example application. Design objects exist independently of an application. They can thus be accessed across applications. The numerous design object types are categorized into eight groups[67]: data objects, process objects, user objects, rule objects, integration objects, group objects, content-management objects, and notification objects. Design objects can be traded on the Appian *AppMarket,* where private developers as well as consulting firms like *KPMG* or *PWC* publish application artefacts.

[65] https://appian.com/platform/overview.html, accessed 09-07-2021

[66] https://appian.com/customer-success/education.html, accessed 12-20-2021

[67] https://docs.appian.com/suite/help/21.3/design-objects.html, accessed 09-08-2021

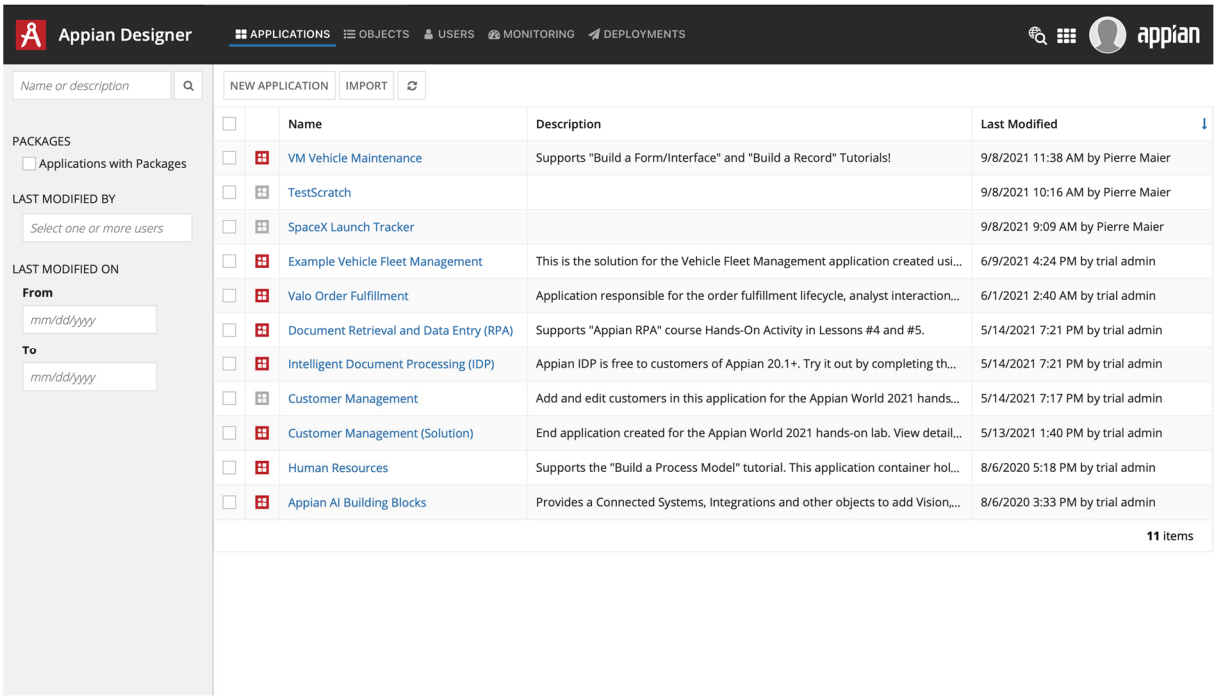

Figure 67: Appian Designer applications overview

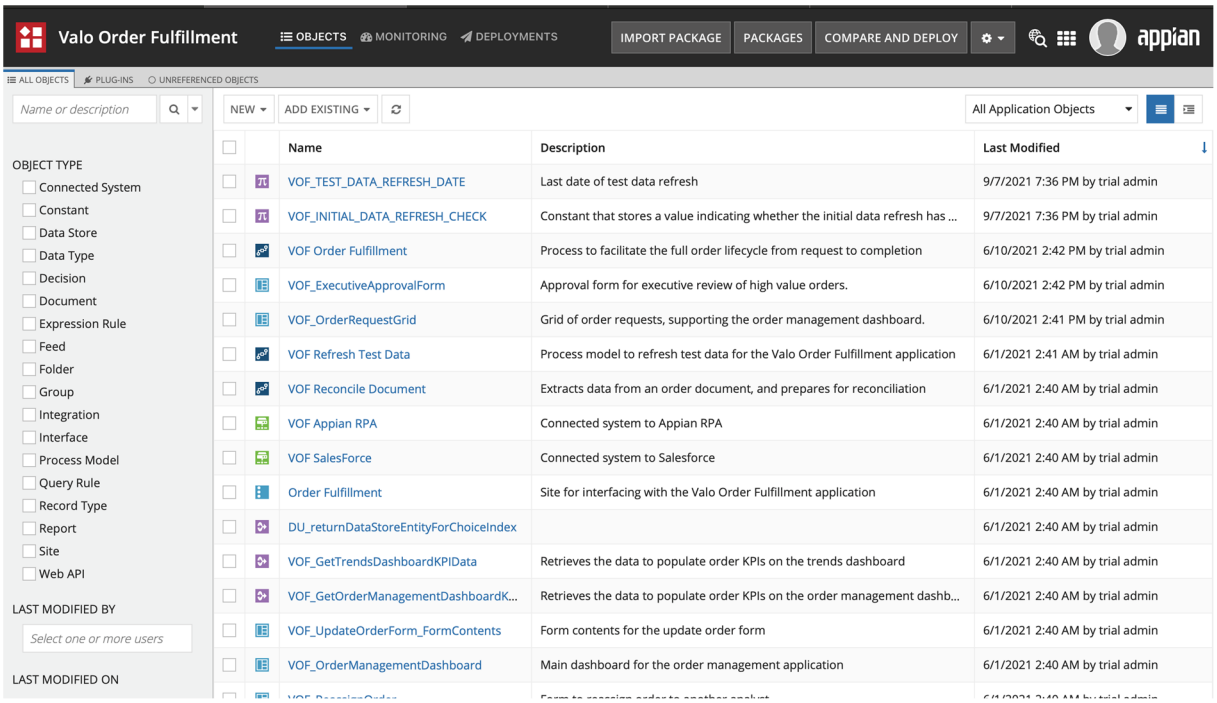

Figure 68: Appian example application overview of design objects

Static Perspective. Three data design object types are available on the Appian platform: “Data Type”, “Data Store”, and “Record Type”. A “Data Type” serves the specification of the attributes of an entity type. Data types like Integer or String are available. In addition, common database constructs like primary and foreign keys can be defined here (see figure 69). “Data

Type" design objects are required to define a "Data Store" design object. A "Data Store" is a collection of user-defined entity types, i.e., "Data Type" design objects. Within a "Data Store" developers can choose to persist the data on the Appian platform or access some external database system. For external database systems an explicit mapping of the entity types must be specified by the user. For internally persisted data, the Appian platform auto-generates a corresponding SQL script, which can be adjusted by the users. The created database tables can be accessed through *phpMyAdmin* in the *Cloud Database* module of the platform. "Record Type" design objects must be defined to connect a GUI page and persisted data (see *GUI Development*). A "Record Type" can access multiple data sources as either defined through a "Data Store" design object or through API calls that send data in JSON format (integrations of external sources are elaborated in *Dynamic Perspective*). These data sources may be adjusted for a single "Record Type" design object. For example, it is possible for users to specify filters (only include data that where condition X is met) or to select only a number of database fields. The resulting data model is also displayed graphically (see figure 70). The modeling language is proprietary and only includes a very limited set of language concepts: data sources are presented as nodes and associations as directed edges. The diagram cannot be edited by the user. Reference data models are not available.

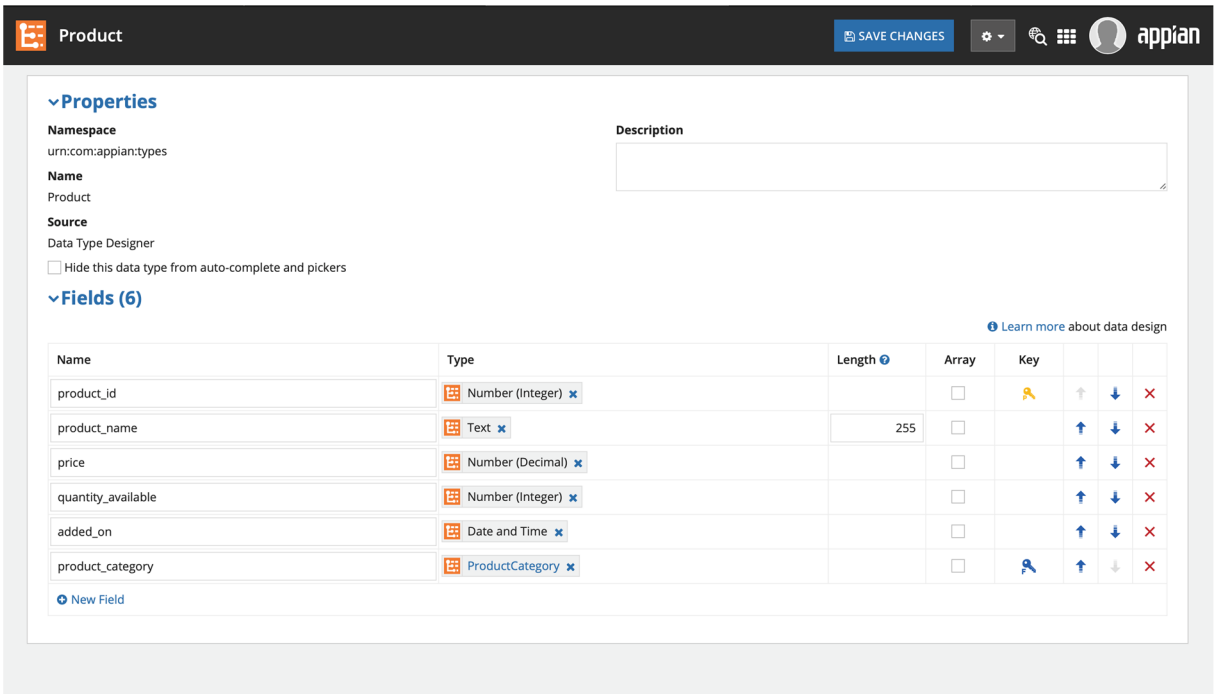

Figure 69: Appian Data Type example

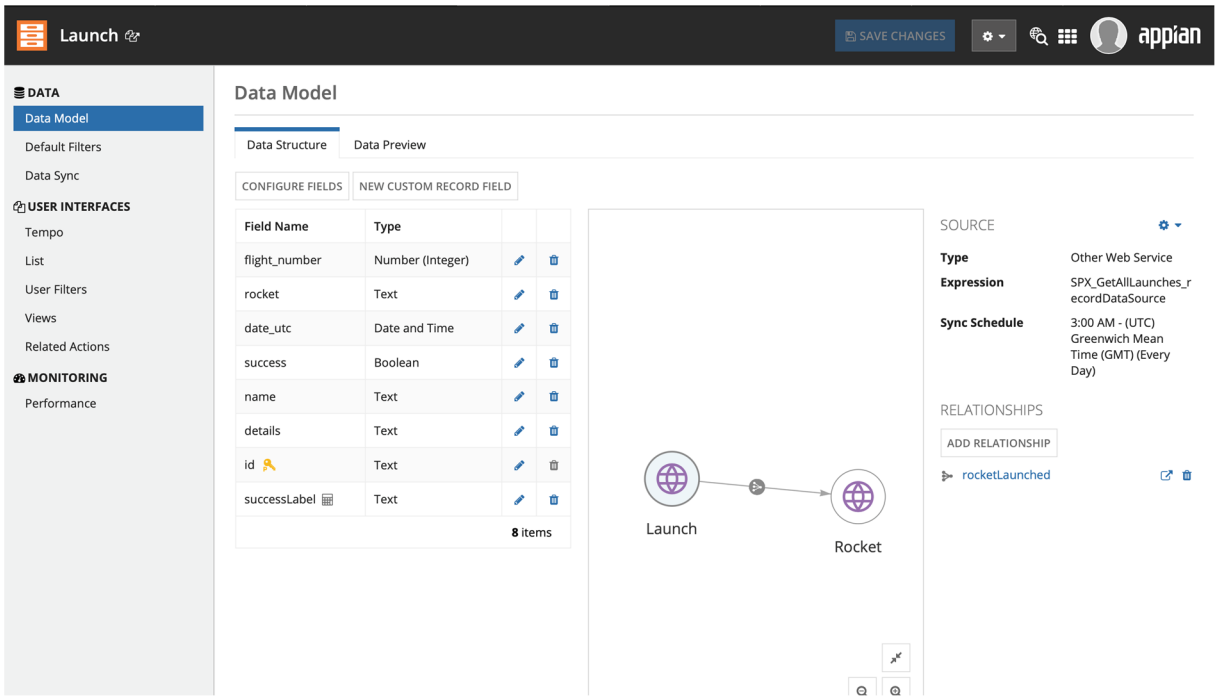

Figure 70: Appian Record Type example

Table 57: Appian static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• "Data Type", "Data Store", and "Record Type" design objects	• proprietary	• diagram viewer as part of "Record Store" design object
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic data types like string and integer are provided • reference data models are not available	• distinction between externally and internally persisted data • relationship and cardinality	• all data sources (internal and external) must comply to a user-specified mapping • internal database can be accessed via phpMyAdmin • generated DDL scripts can be adjusted
Focus on Integration		
Common static abstractions across a range of applications	Since design objects are generally available in all Appian applications, the use of shared static abstractions is supported.	
Access to common data repositories across a range of applications	It is possible to access a common data repository across applications. Whether this is actually done, depends on the application developer.	

Functional Perspective. Custom functions can be specified via “smart services” that are part of workflow models (see *Dynamic Perspective*), “Decision” design objects, and “Expression Rules” design objects. Smart services represent system-defined functions for generic business operations or platform configuration, e.g., to create an Excel report or adjust existing file structures. “Decision” design objects serve to specify multi-criteria conditions resulting in a variety of different potential responses (see figure 71). They basically represent common switch-case statements. More elaborate functions beyond conditional statements can be implemented via “expressions”. Expressions are small scripts written in a proprietary programming language. Such expressions can be embedded in workflow models and GUI pages. “Expression Rule” design objects serve to define shared expressions to avoid functional redundancy (see figure 72). Appian provides numerous reference functions to embed in these expressions, ranging from functions to manipulate data structures to arithmetic calculations and platform-specific adjustments (e.g., to retrieve a list of active users).

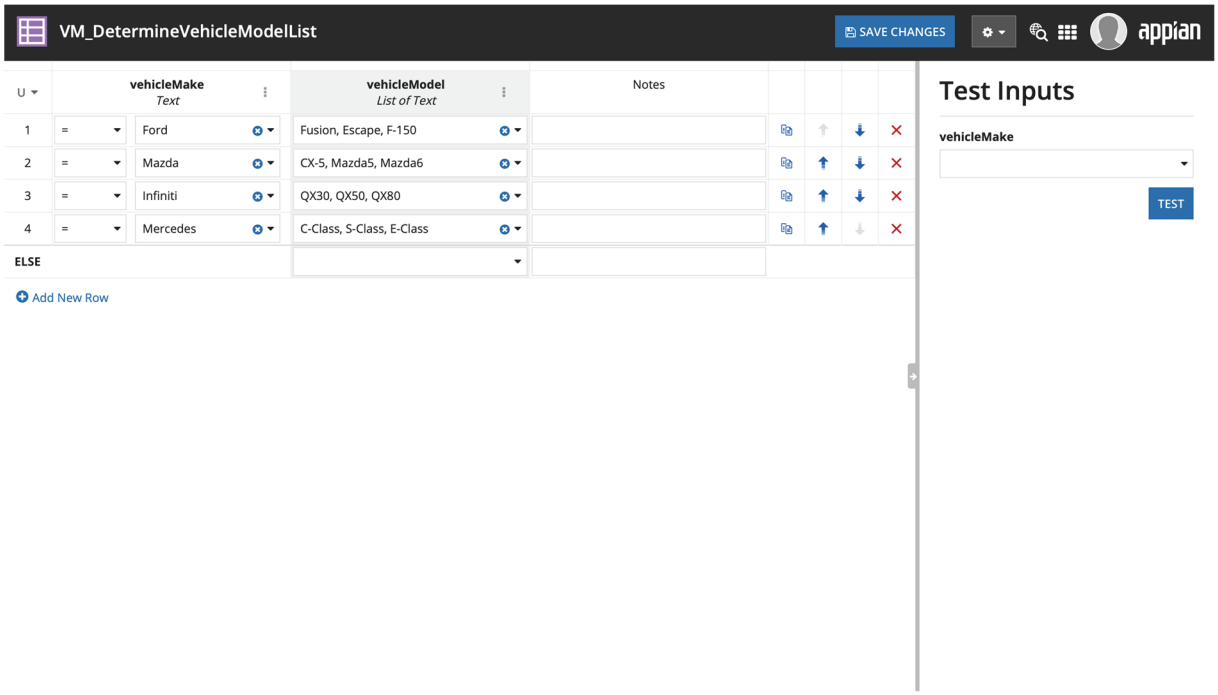

Figure 71: Appian "Decision" design object example

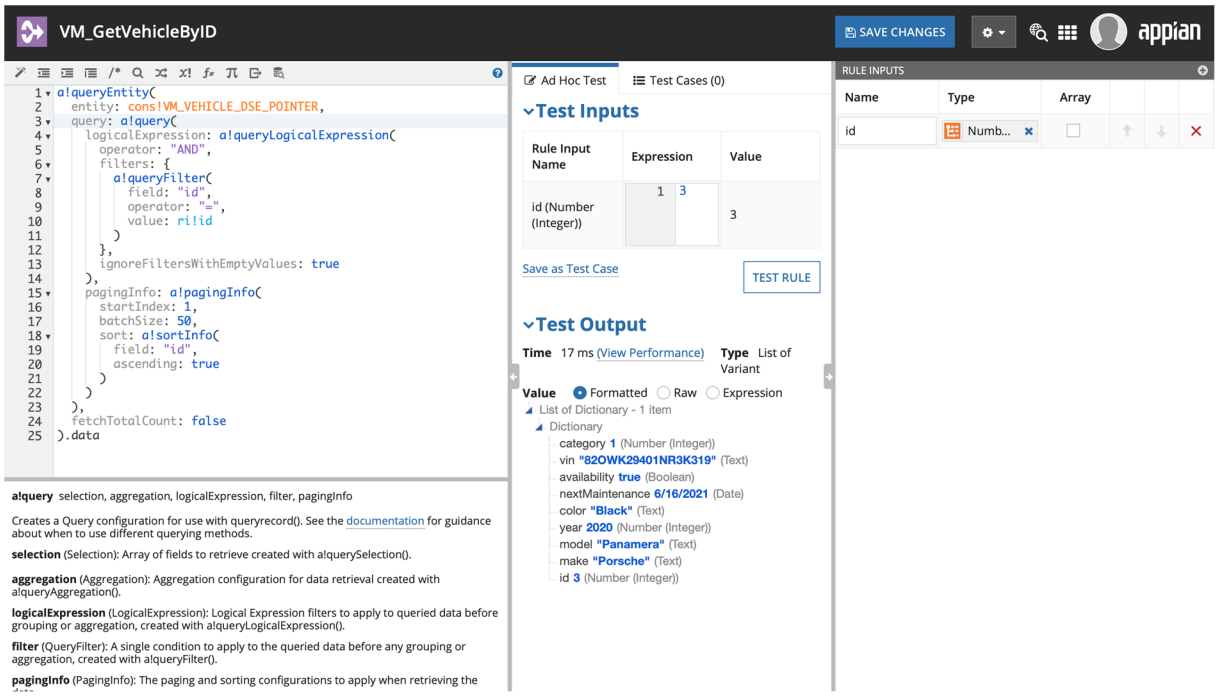

Figure 72: Appian "Expression Rule" design object example

Table 58: Appian functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• no conceptual representation beyond "Decision" and "Expression Rule" design objects	• *inaccessible*	• built-in source code editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic reference functions for simple arithmetic and conditional functions are provided"	• composition	• source code of expressions can be accessed

Dynamic Perspective. Appian provides a separate workflow modeling component through "Process Model" design objects (see figure 73). The modeling language is based on BPMN but provides some additional task types – the so-called "smart services" which are explained in the *Functional Perspective* analysis category. Integrations with external systems are supported through the design objects "Connected System", "Integration", and "Web API". A "Connected System" is closely intertwined with an "Integration". A "Connected System" design object

serves to specify connection information to an external source either through HTTP or Open-API calls. Some connections to access external database systems and AI services are pre-implemented by Appian (see *Further Aspects*). An "Integration" design object specifies the interaction with a "Connected System" (e.g., send a JSON request or access some file on the connected system). "Web API" design objects serve to define access to web services.

It is also possible to define, manage, and execute RPA bots in the separate *Appian RPA* module. RPA bots mimic simple user actions and can be designed based on several available templates. The execution of RPA bots requires a local machine. The sequence of actions can be arranged graphically (see figure 74). The respective Java code is generated automatically through the recording of user actions either via Appian or other plug-ins.

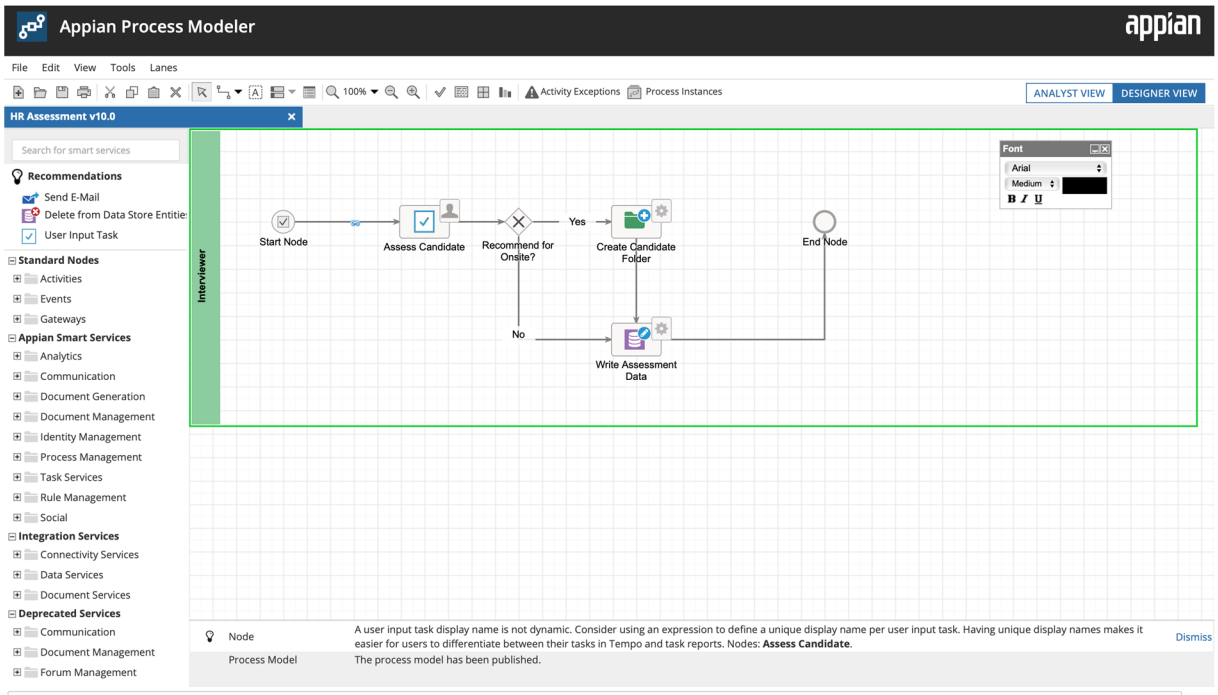

Figure 73: Appian Process Modeler

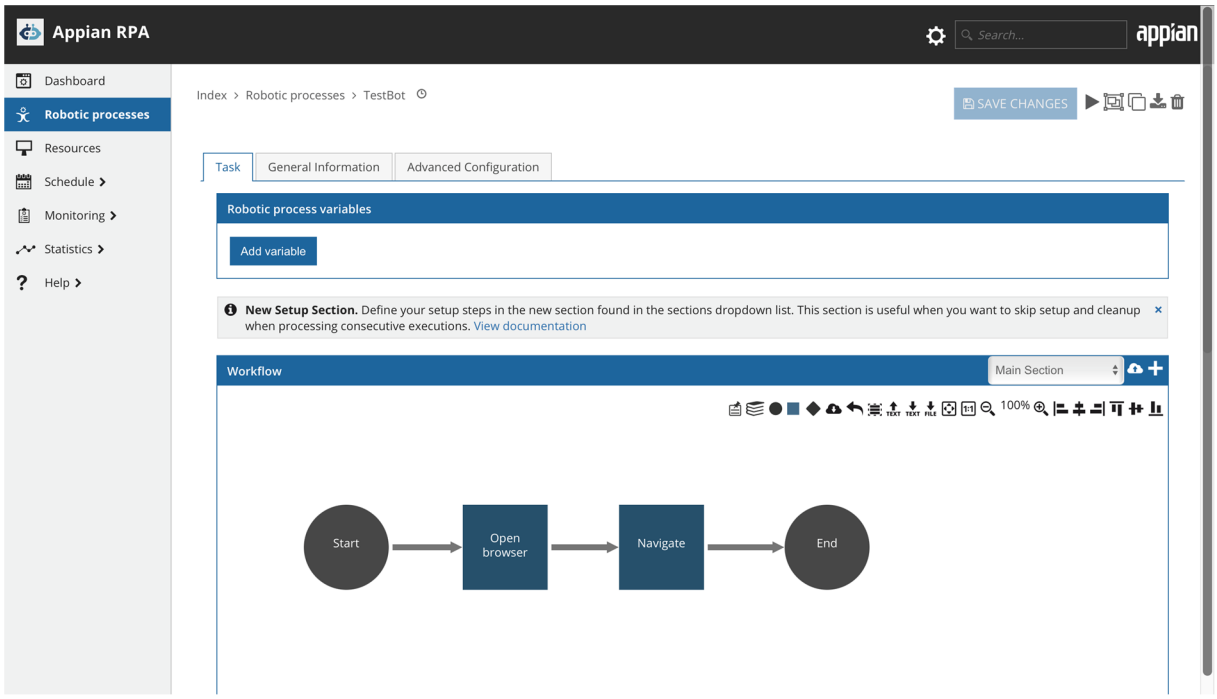

Figure 74: Appian RPA "browser" template

Table 59: Appian dynamic perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• business process models • generic process models (for RPA bots)	• BPMN • proprietary (for RPA bots)	• model editor • for RPA bots: Java code generator
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• generic business process elements as part of BPMN specification • "smart services" convey pre-implemented functions for workflows	• Language concepts of BPMN	• Java source code can be accessed for RPA bots • Implementation-level documents for workflows cannot be accessed

GUI Development. To develop GUIs, four design objects are offered within the Appian platform: "Report", "Process Report", "Interface", and "Site". "Report" design objects can serve to display persisted data as specified in "Record Type" design objects. "Process Report" design objects display data from running workflow instances. Such reports are not inherently part of

the GUI of a developed application, but can be accessed in the separate *Tempo* module. "Interface" design objects are used to specify a mapping between user-defined entity types to GUI elements. Interfaces are designed through a drag-and-drop editor with several generic system-defined GUI elements (see figure 75). The GUI is synchronized with a proprietary "expressions" code that can be accessed in the "Expression Mode" of an "Interface" design object. A "Site" design object comprises a GUI environment for deployed applications. Each user can be provided with a different "Site". These "Site" objects consist of a navigation bar and different GUI pages, whereby each GUI page is, directly or indirectly, constituted through an "Interface" design object. GUIs can also be generated automatically based on a data model as defined in a "Record Type" design object or through Appian's *Quick App Designer* module, where the required design objects for an application are automatically created based on the specification of a single entity type (see figure 76).

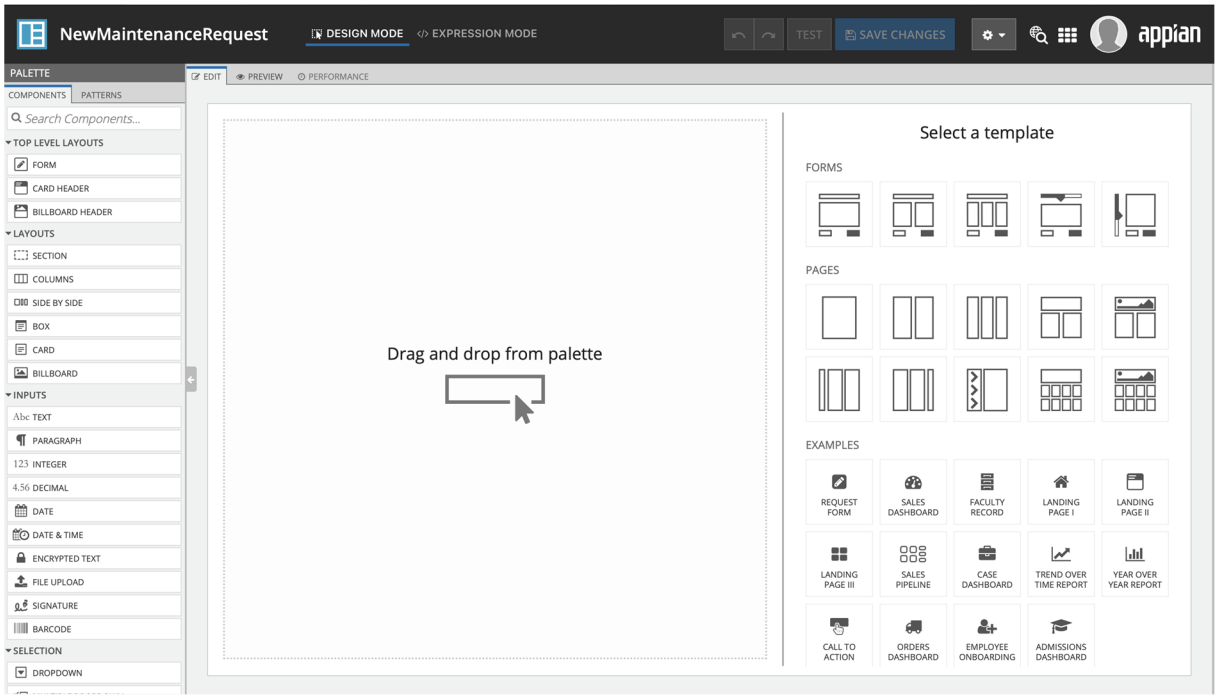

Figure 75: Appian "Interface" design object example

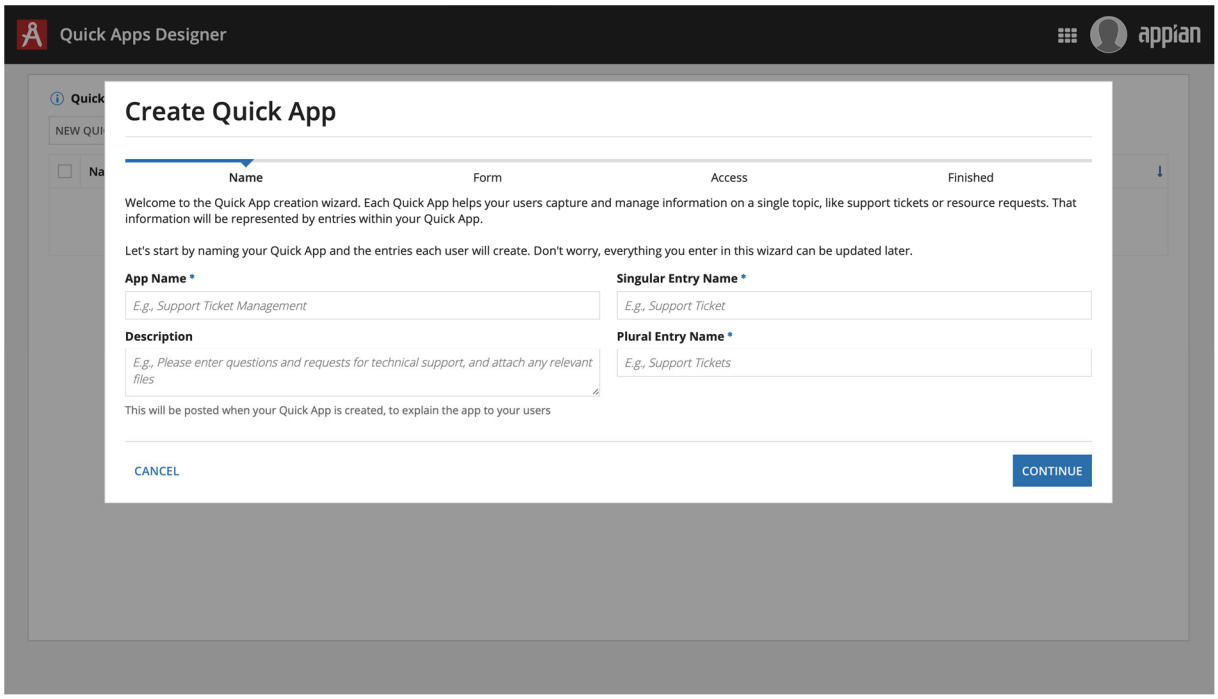

Figure 76: Appian Quick Apps Designer module

Table 60: Appian GUI development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*
• GUI models (mostly in "Interface" design objects)	• drag-and-drop GUI Editor	Implementation of MVC pattern
Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• library of general-purpose GUI elements • no further domain-specific reference abstractions	• GUI elements can be adjusted accordingly	

Further Aspects. Applications are deployed on an Appian web server and can be accessed via a regular web browser. A list of all executed deployments can be viewed in a separate tab for each application. Entire applications can also be exported as a set of XML. The exported files do not directly support any deployment independent of Appian.

Appian does not offer any methodical application development support. "Notification" design objects can be defined which are used within workflow models to display certain messages for Appian users. The use of applications is guided by a definition of user groups

through a respective design object. Users can be assigned to multiple groups. Every design object within Appian needs to be provided with CRUD rights for selected groups. Groups can also be arranged in group types, although these do not include any inheritance of rights.

The only AI capability provided directly within Appian is the "next best action" of the workflow modeler. This feature is also displayed in figure 73, where a list of "recommendations" can be seen in the top left corner. The feature might be based on an inductive ML model. However, neither the model type nor its data foundation is accessible. The lack of explanation makes the use of the predictions hardly assessable. The self-designated "Appian AI" component of the platform emphasizes and supports the integration of external AI services as provided by the Google Cloud Platform. To access these feature user credentials for a Google Cloud Platform domain must be provided.

Table 61: Appian further aspects summary

<table>
<tr><td colspan="2">Accessibility and Convenience of Use</td></tr>
<tr><td colspan="2">• no methodical development support
• the vast number of available design objects and confusing terminology present a burden to the initial use of the platform
  ○ the "Quick Apps Designer" module is insufficient to overcome this limitation, since its features are too restricted
  ○ Appian is thus hardly adaptable to a diverse set of user groups and demands some experience in application development
• data modeling language lacks expressiveness</td></tr>
<tr><td colspan="2">Modes of Use</td></tr>
<tr><td>Platform (i.e., Application Development)</td><td>Application</td></tr>
<tr><td>• users of Appian must be assigned to some user group
• each design object must specify the required CRUD rights for all user groups
• user notifications can be embedded in workflows
• methodical application development support is not provided</td><td>• different users can be provided with different GUI screens
• no inherent support for collaborative use of an Appian application</td></tr>
<tr><td colspan="2">Focus on Deployment and Scalability</td></tr>
<tr><td>Target Location</td><td>Support</td></tr>
<tr><td>• cloud-based platform, accessible via a regular web browser</td><td>• support to deploy Appian applications in Dockers containers</td></tr>
<tr><td colspan="2">Focus on Artificial Intelligence</td></tr>
<tr><td colspan="2">• "next-best action" in workflow models not convincing
• support for integration of AI services as offered by Google Cloud Platform</td></tr>
</table>

5.4.2.3 Appian: Conclusion

Emphasized Areas of Application Development. The Appian platform does exclusively focus single areas of application development. Dynamic and functional aspects as well as the configuration of GUI pages are most sophisticated features of the LCP. This includes the workflow modeling component and the design objects for functional specification. Static features are to some degree limited in that visual data modeling is not supported. Increase of development productivity is mainly realized through the provision of design objects. These design objects are application-independent, which can support reuse and integration.

Provision of Abstractions. The comprehensive set of features of the platform attempt to fade out implementation-related details. However, no higher-level abstractions are presented for this purpose and any specification of the available design objects demand some familiarity with the respective area of application development. Configuration of the internal database through *phpMyAdmin* can serve as a particularly illustrative example here. The provided representations are hardly accessible by lay developers. Domain-specific abstractions, beyond solutions that are available on the Appian *AppMarket*, are not available. Features to support collaborative development between lay and professional developers could not be identified either.

Role of IT Professionals. The use of the platform is generally demanding and strongly focusses a modular approach in software development. The Appian platform relies on a vast number of interrelated design objects. Except for the *Quick Apps Designer* module, lay developers are likely to be overburdened by this complexity. However, applications developed with the *Quick Apps Designer* do not include any workflows or custom function and rely on one entity type only. Such applications are thus not sufficient for most use cases without further adjustments by professional developers.

5.4.3 Pega Platform

The Pega Platform is offered by Pega, a software company that has existed for around 40 years. It is based in Cambridge, MA, and runs about 40 offices on all continents except for Africa.

5.4.3.1 Pega Platform: Profile of Vendor

Product Portfolio. Pegasystems (regularly abbreviated as Pega) offers a portfolio of different, interacting products jointly referred to as *Pega Infinity*.[68] The products represent pre-implemented software solutions tailored for specific needs, e.g., customer service or sales automation. Its low-code product is the *Pega Platform*.

[68] https://www.pega.com/infinity, accessed 09-17-2021

Product Provenance. The company Pegasystems emerged in 1983.[69] The earliest archived web entry of pegasystems.com can be identified on Nov 13, 1996, where aspects such as "workflow solutions" and "automation" are marketed. The Pega Platform is said to empower businesses through "smart business process management" (Sep 26, 2004). Business process management remains a primary focus of marketing in the following years. The term low-code is mentioned since Oct 02, 2018, and the Pega Platform is referred to as a low-code platform at least since Sep 30, 2019.

Focus on Marketing. Low-code development through the Pega Platform is advertised to "empower those without tech backgrounds" through "visual tools and drag-and-drop functionalities. [70] Lay developers and professional developers are addressed. Further concepts like BPM, AI, application testing, and UX design are also part of their current marketing message.

Table 62: Pega Platform profile of vendor summary

Product Portfolio
• wide variety of software products that address specific needs • the Pega LCP serves as the main application development product
Product Provenance
• company emerged in 1983 • early focus on workflows and business process management • focus shifted towards application development • labeled LCP since late 2019
Focus on Marketing
• low-code to support lay developers • both kinds of developers are explicitly addressed • wide variety of features are advertised

5.4.3.2 Pega Platform: Analysis of Platform Features

The Pega Platform is divided in four workspaces. These are denoted as *App Studio*, *Dev Studio*, *Prediction Studio*, and *Admin Studio*. Only the former two serve directly to develop user-facing applications. *App Studio* is the more user-friendly development environment oriented towards

[69] https://www.pega.com/system/files/resources/2021-03/pega-corporate-factsheet-1-30-2020.pdf, accessed 09-17-2021

[70] https://www.pega.com/products/platform/low-code-app-development, accessed 09-17-2021

business users. *Dev Studio* provides access to a more extensive set of features to develop applications. *Prediction Studio* is used to create and manage ML models. *Admin Studio* is not included in the trial version and is thus not considered in the subsequent analysis. The *App Studio* home screen, which provides the overview of an example application is displayed in figure 77. The navigation pane on the left highlights the six components included in *App Studio*: "Overview", "Case Types", "Data", "Channels", "Users", and "Settings". Unless otherwise noted, the subsequent analysis refers to features provided in *App Studio*.

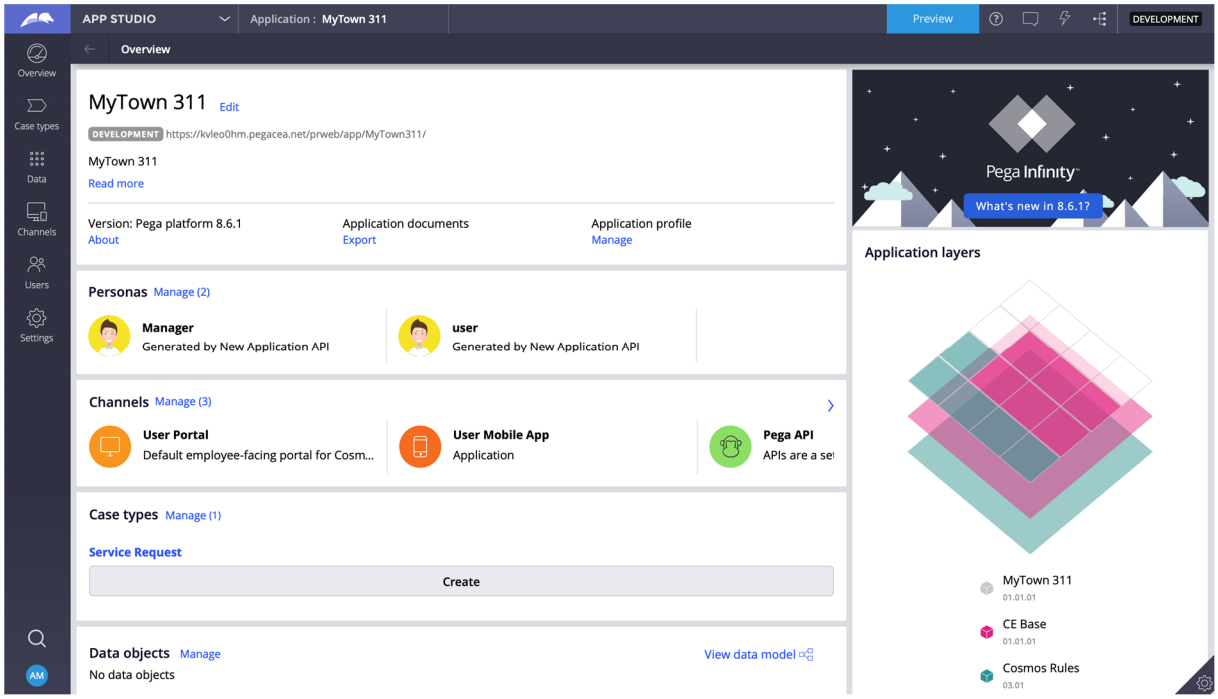

Figure 77: Pega Platform App Studio example application overview

Static Perspective. The definition of entity types is different for the two development environments. In *App Studio*, entity types are specified through "data objects". Through data objects, users can define the attributes of an entity type and additional validation rules. General data types like "text" or some business-generic data types like "phone" are offered for this purpose. Data objects also contain "views" (no GUI element, but a selection of table fields – similar to a database view) and "pages". Data pages are used to define operations on the underlying data. The definition of a data object in App Studio is guided by auto-generated data pages for standard read and write operations. Further implementation-related details, like the definition of database keys or inheritance relations, is only possibly in *Dev Studio*. SQL queries cannot be accessed by the user in neither environment. Data can be persisted on the Pega platform (the exact database technology is not accessible) or some connected external source. Mappings between entity types can be defined through "data transforms" in *App Studio*, which is relevant

for any transient-persistent as well as external-internal source mapping. Two different representations of the underlying data model are available: a "Visual Data Model" and an "Integration Map". The Visual Data Model uses a proprietary modeling language to merely display case types and their use of data objects (see figure 78). These diagrams can be accessed in *App Studio* and *Dev Studio*. "Integration Maps" are only available in *App* Studio. These highlight the sources of user-defined entity types (see figure 79). Reference data models are not available. The *Pega Marketplace*[71], however, allows to download components or entire applications at a certain fee.

[71] https://community.pega.com/marketplace/search, accessed 10-13-2021

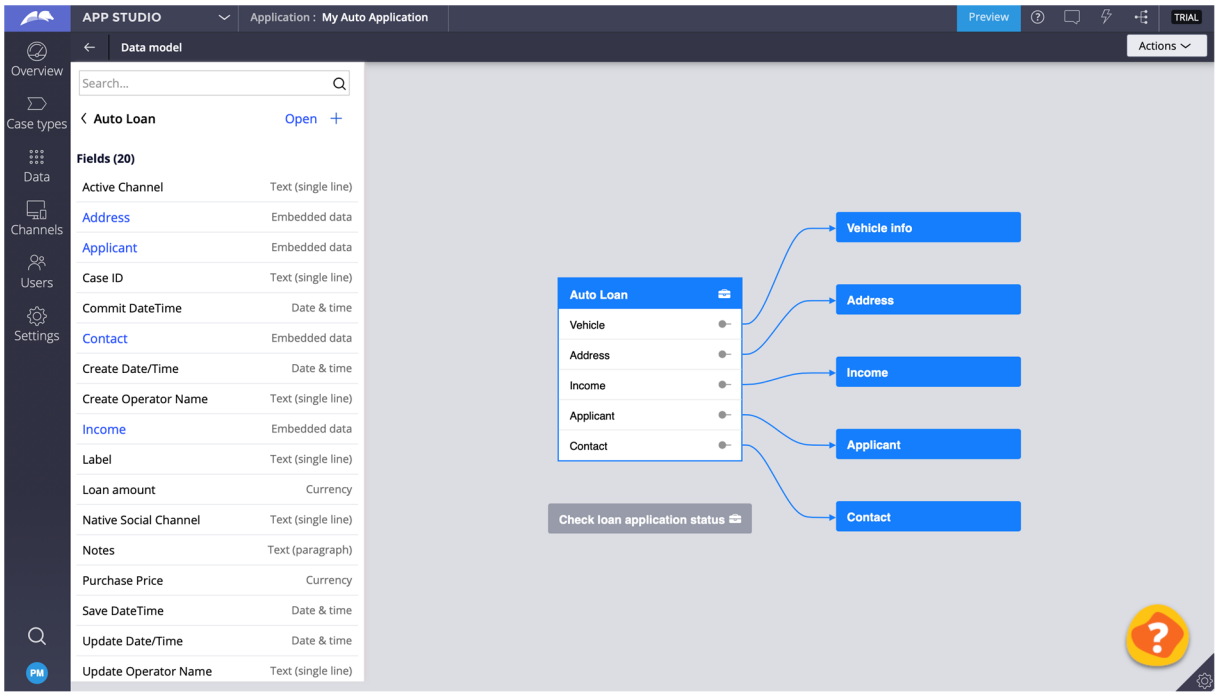

Figure 78: Pega Platform App Studio Visual Data Model example

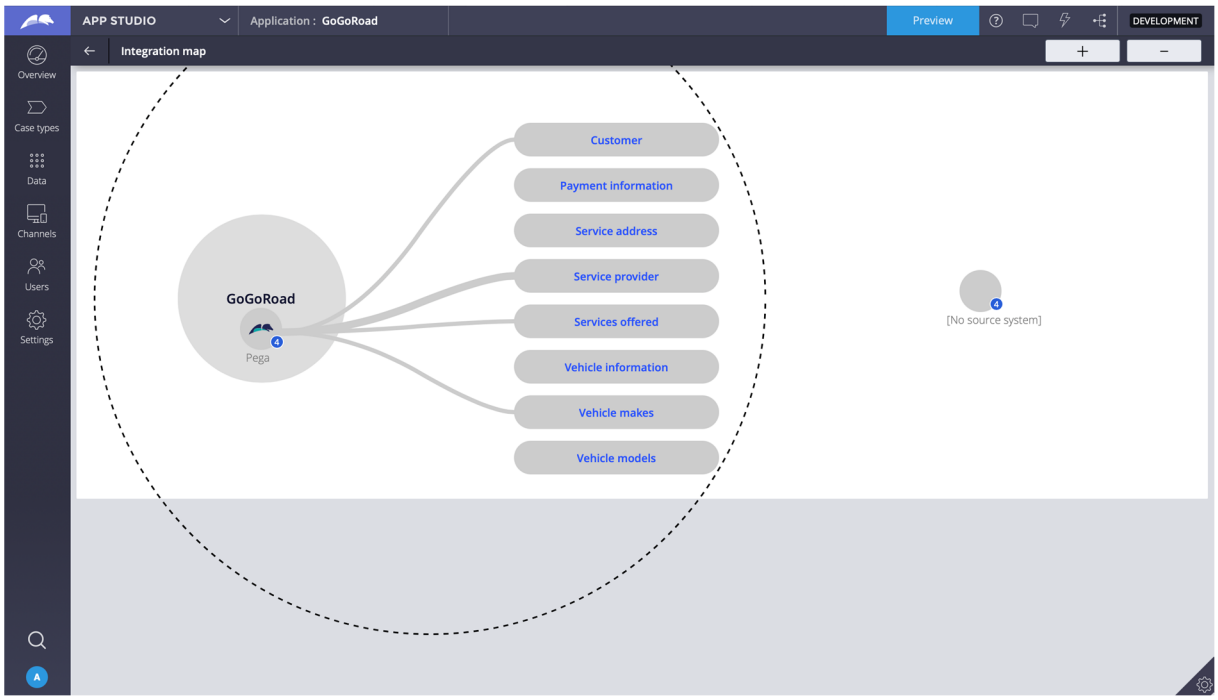

Figure 79: Pega Platform App Studio Integration Map example

Table 63: Pega Platform static perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• "Integration Map" • "Visual Data Model"	• both representations follow a proprietary language	• diagram editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• some generic data types like text and integer as well as some composite data types like phone and location are offered • no further reference data models are offered	• inheritance (in *Dev Studio*) • data source (in Integration Map)	• implementation-level documents are not available
Focus on Integration		
Common static abstractions across a range of applications	Applications developed within Pega can access the same static abstractions.	
Access to common data repositories across a range of applications	It is possible to persist all data in an internal Pega database.	

Functional Perspective. The specification of entity types in *App Studio* can include attributes, whose values are calculated based on some user-defined function. For this purpose, users can define either a decision table (multi-conditional statements), a function (common pre-written iterative functions like sum, average, or max), or an expression (spreadsheet-like arithmetic or conditional statements). Other than this and the execution of workflows, the Pega Platform includes an extensive catalog of "rule" types that can be accessed in *Dev Studio*. Each rule type provides a user with web forms to support either the specification of particular functions or configure certain properties of an application. Rule types are categorized in 15 different groups (e.g., "Integration-Mapping", "Process", "Technical", or "Security") with up to 31 rule types each.

Every user- or system-define rule is assigned to a "context" and a "ruleset". Rulesets are a collection of rules. Contexts are arranged hierarchically and allow to assign a layer to a ruleset for reusability. Rulesets that are assigned to a higher-level context are inherited to all lower-

level contexts. Upon creation, every Pega application must be assigned to a context. The context, then, determines which set of pre-implemented rules can be used within an application.

Examples for rule types to specify custom functions are “decision tree”, “decision table”, and “function”. Decision tables are multi-conditional return operations arranged in tabular form (see figure 80). Completeness and consistency are validated automatically. Decision trees define if-else statements based on different attributes. “Function” rules are specified through a built-in Java source code editor. Rules can be embedded in the execution of workflows (see *Dynamic Perspective*).

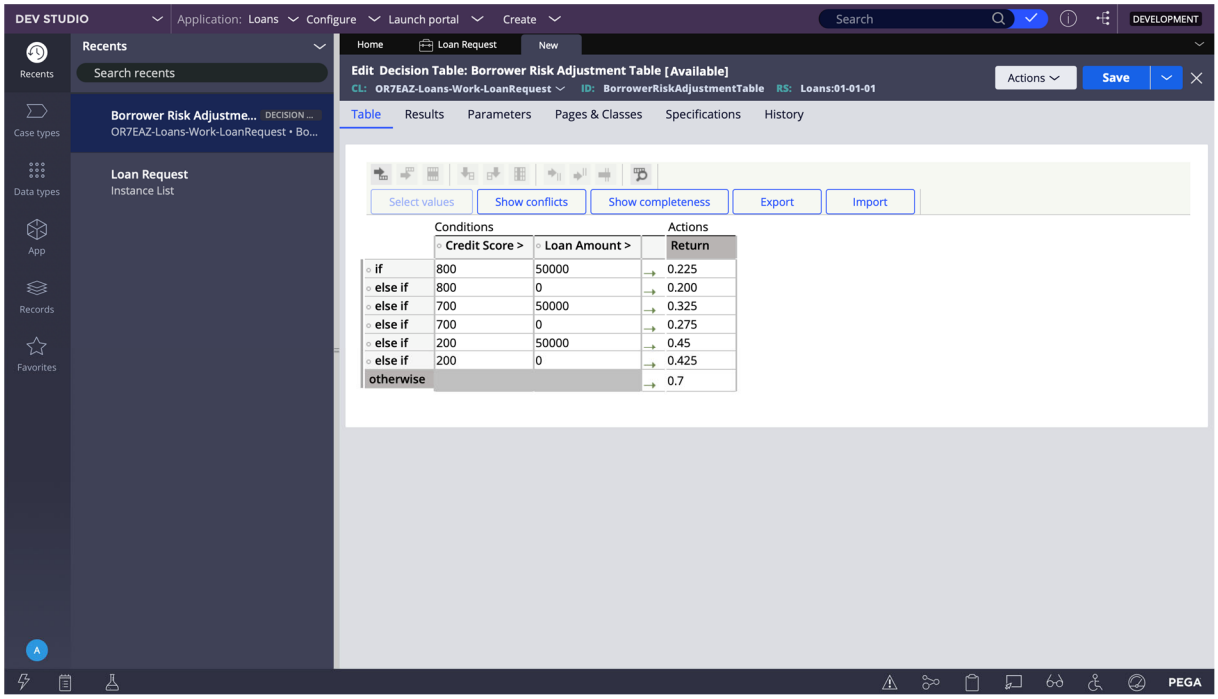

Figure 80: Pega Platform Dev Studio decision table

Table 64: Pega Platform functional perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• no diagram representation of functions • “decision tables”	*not accessible*	• various GUI screens that demand pre-defined user input in *Dev Studio* • Java source code editor
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• library of reference functions provided by Pega	• depends on rule type	• Java source code

Dynamic Perspective. Workflow types are referred to as "case types" within the Pega Platform. An example case type in *App Studio* is illustrated in figure 81. Each case type consists of different "stages", at least one start and one end stage underlined green and red respectively. Each stage contains a set of "processes", whereby each process serves to group some number of "steps". "Processes" within the Pega Platform only allow for a sequential definition of steps. No further differentiation of flows or gateways is included. Steps are used to execute a diverse set of functions. Three kinds of steps can be embedded in case types: "processes", "user actions", and "automations". Largely self-explanatory, "processes" refer to some predefined process, itself a collection of further steps (e.g., "Resolve Issue" in figure 81 would be a process that can be referenced in further case types), "user actions" correspond to manual tasks requiring user input, and "automations" are automatically executed tasks that require no additional user input like "generate PDF" or "persist case". *App Studio* does not allow for adding user-defined functions. For stages, processes, and steps a "service-level agreement" (SLA) can be added. Every stage, process, and step has a system-defined "urgency" attribute with an integer value. SLAs can be defined to update this urgency attribute based on a user-specified run duration. Data, "Personas" (see *Further Aspects*), and "Channels" (see *GUI Development*) can be associated to stages. Stages are generally displayed from left to right. Alternative case paths, or cases that do not fit on a single line, are continued underneath. More extensive features to configure case types are available in *Dev Studio*. Here, it is also possible to access and edit process diagrams, i.e., the collection of steps as defined in *App Studio* (see figure 82). This allows, inter alia, to define custom "flow" actions, which are executed before or after a step. The auto-generated Java and XML code can also be accessed. Only in the process editor in *Dev Studio*, additional "utility" tasks can be added which are not available in *App Studio*. Utility tasks can execute "rules" (see *Functional Perspective*).

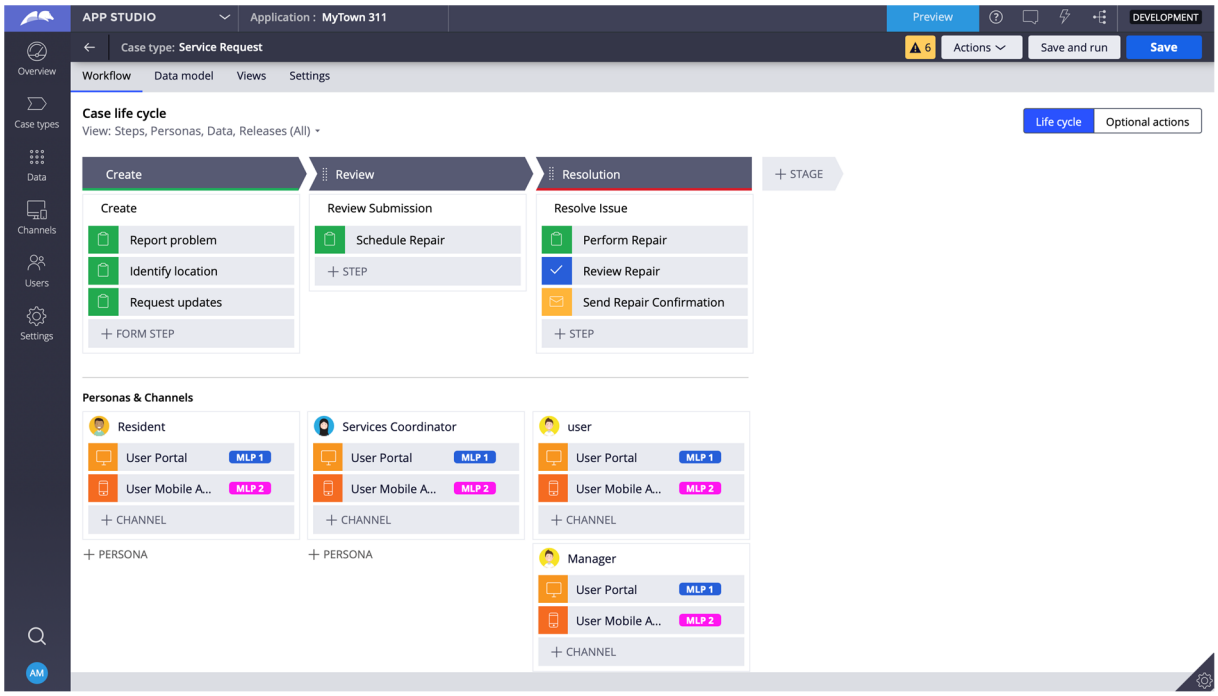

Figure 81: Pega Platform App Studio case type example

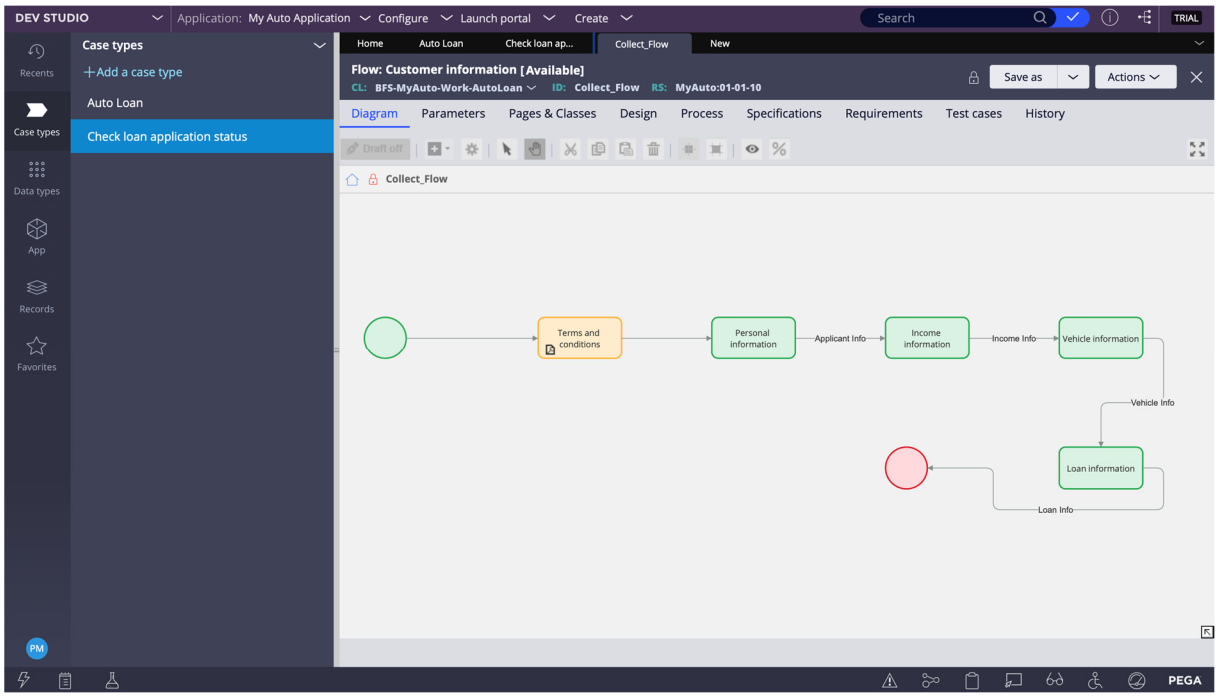

Figure 82: Pega Platform Dev Studio process diagram

Table 65: Pega Platform dynamic perspective summary

Focus on Representations		
Conceptual representations	*Languages*	*Components*
• generic process models	• proprietary	• diagram editor for processes • XML and Java code generator for process diagrams
Focus on Reuse and Adaptability		
Reference abstractions	*Language concepts*	*Access to external sources and to implementation level documents*
• reference abstractions are only available as part of entire applications	• composition • "process", "automation", "user action" task types • "case type" models are disseminated into distinct stages	• workflow schema in XML for processes • Java source code for processes

GUI Development. UIs of applications developed with the Pega Platform are specified through "channels". Channels can be messaging services (e.g., send email to execute workflow instance) or common GUIs. Three kind of GUI channels are distinguished: "mobile", "web mashup", and "portal". "Mobile" channels are used to configure native mobile apps that can be deployed on iOS and Android devices. Mashup channels generate HTML frames that can be embedded in some external web site. Portal channels are responsive web GUIs, where the GUI elements adjust to the respective display size. Each GUI consists of two kinds of GUI pages – user-generated ("landing page") and system-generated ("custom page"). Custom pages include, among others, a "Home", "Reports", and "Explore Data" GUI page as well as pages to create workflow instances. The configuration of the arrangement of GUI pages is displayed in figure 83. Landing pages can be developed in a drag-and-drop editor (see figure 84). Each GUI page consists of "sections", i.e., layout schemas, and "widgets", that are placed within a respective section. Pega offers several general templates for sections and widgets. Separate, domain-specific reference GUIs are not available. By default, the Pega Platform generates one portal and mobile channel automatically. In *Dev Studio*, user-defined templates can be designed either per drag-and-drop or via the specification of HTML code.

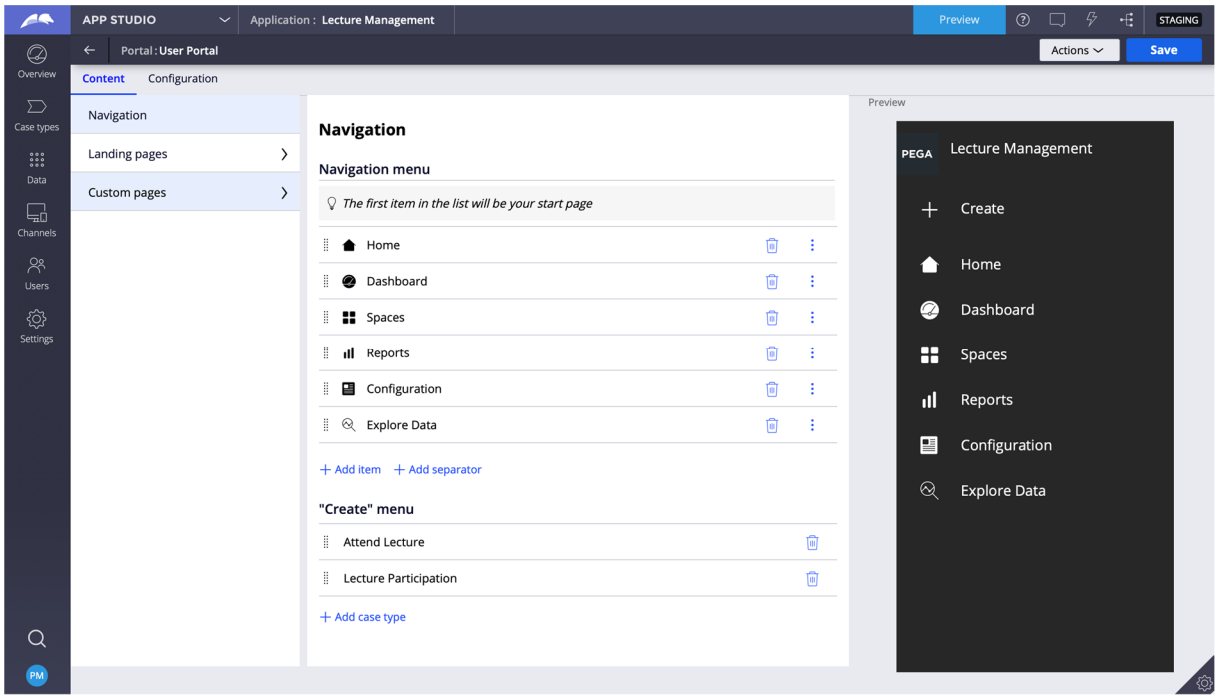

Figure 83: Pega Platform App Studio GUI portal configuration

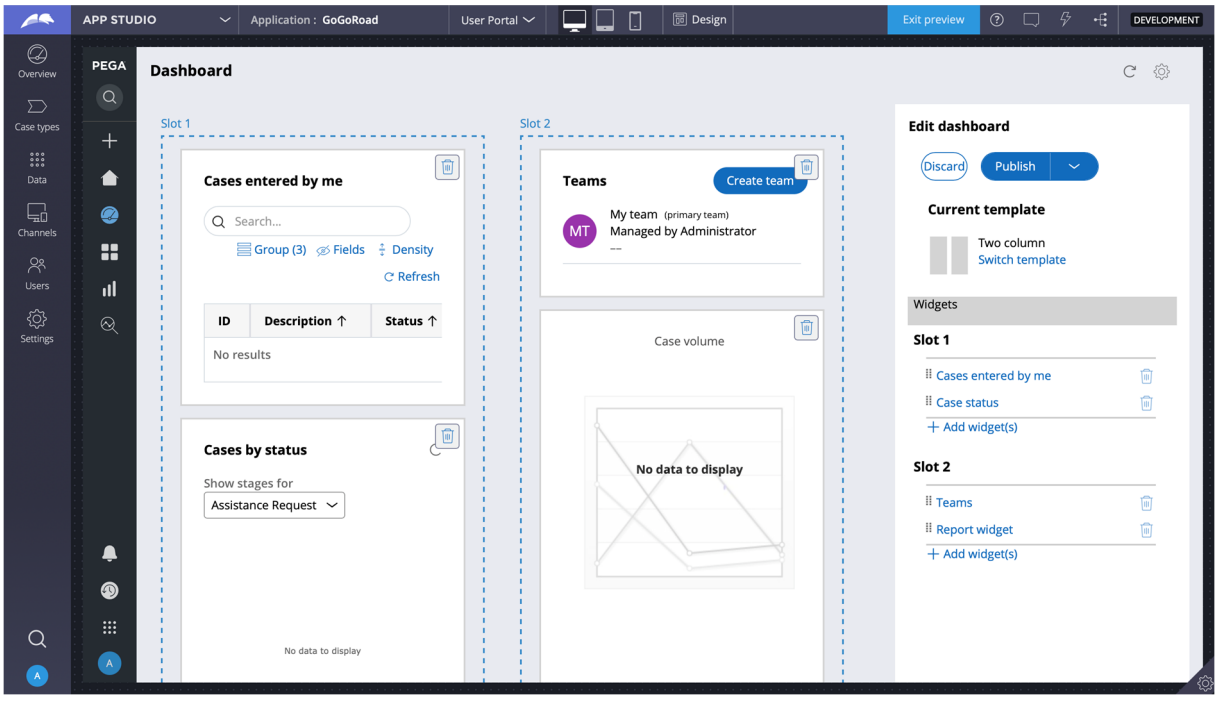

Figure 84: Pega Platform App Studio GUI designer

Table 66: Pega Platform GUI development summary

User Interaction Perspective		
Conceptual representations	*Components*	*Architectural Aspects*
• GUI model	• drag-and-drop GUI Editor	implementation of MVC pattern
Focus on Reuse and Adaptability		
Reference abstractions	*Adaptability*	
• library of general-purpose GUI elements • reference GUIs are not available beyond the Pega marketplace	• widgets can be adjusted according to their size, position, and style	

Further Aspects. Pega applications are deployed on a web server and can be accessed through a regular web browser. The Pega platform also uses a version control system. Applications can also be designed for native mobile OS support and deployed on the respective devices. Pega applications can also be accessed externally through system-defined APIs. It appears that applications developed with Pega cannot be exported to and executed in other environments. It is merely possible to export an "overview document", which lists all artefacts specified in App Studio (e.g., entity and workflow types).

Three aspects provide explicit support for collaborative development in Pega: the version control system, the "agile workbench", and the "developer collaboration". The agile workbench comprises a list of user stories, bugs, and feedbacks (see figure 85). Corresponding items include common elements like name, description, associated feature (from the application), attachments, and some priority ranking. The "developer collaboration" panel does not offer any specific structure and can simply be used to enter comments visible to all or some specified subset of users. While the "developer collaboration" panel is only visible in *App Studio*, the agile workbench can also be accessed in *Prediction Studio*. A separate "collaboration center"[72] offers a Pega-specific forum for further discussions. End users of the application are assigned "personas". A persona represents a prototypical user category which is assigned to a portal (see *GUI development*), mobile GUI , and a set of CRUD rights for interaction with the various parts of an application (i.e., single GUI pages, entity types, and cases). More detailed configuration options, also to specify developer access, are offered in *Dev Studio*.

[72] https://collaborate.pega.com, accessed 10-13-2021

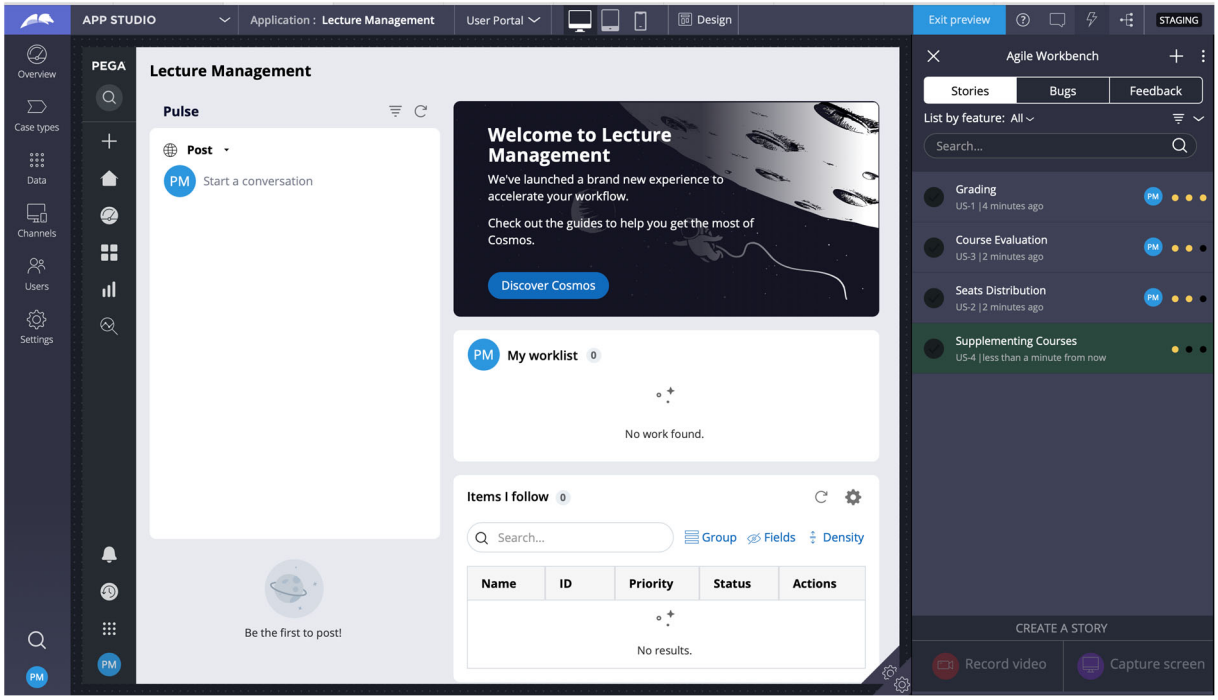

Figure 85: Pega Platform App Studio Agile Workbench

AI services are part of the *Prediction Studio* workspace. Four different kinds of AI models can be specified: “adaptive model”, “predictive model”, “text extraction”, and “text categorization”. Some generic, pre-trained models for text categorization and text extraction are offered by Pega (see figure 86). Predictive models based on Pega ML demand some template for specification, none of which is part of the trial version used for our analysis. Adaptive models demand the specification of dependent (“outcomes”) and independent (“predictors”) variables – the exact model underneath is inaccessible though. Text models can be deductive (rule-based or keyword compliance) or inductive (pattern based on provided data as derived from inaccessible ML model). External access to Google AI and Amazon SageMaker is supported.

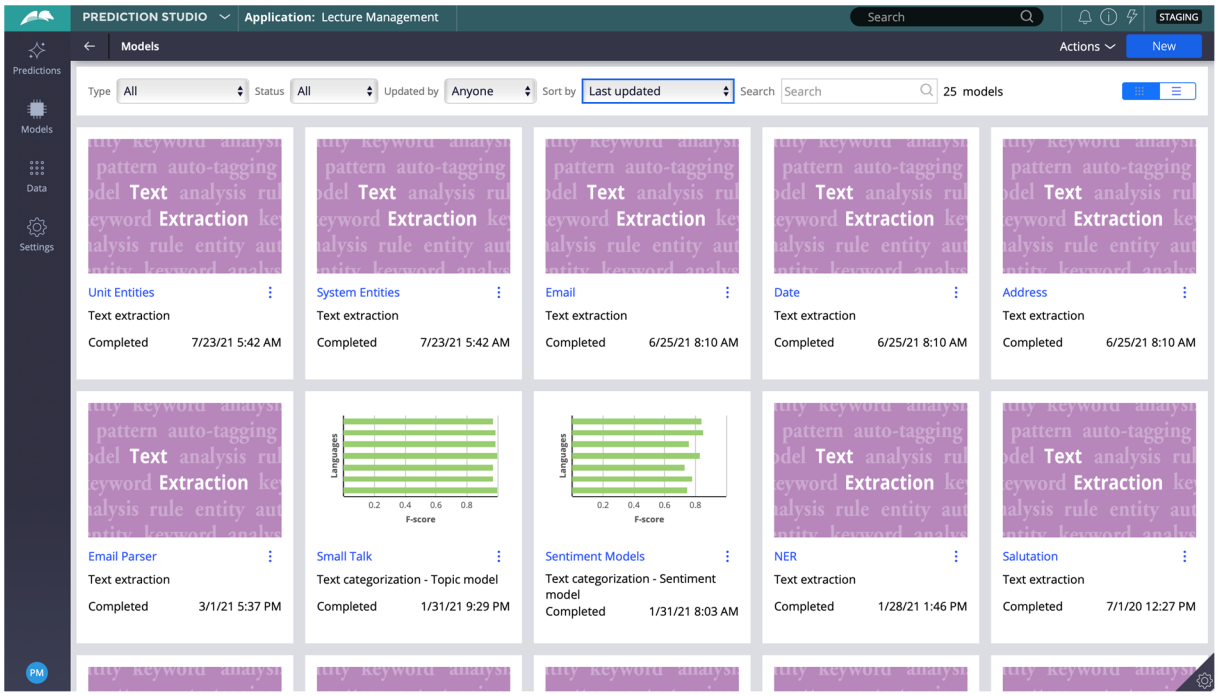

Figure 86: Pega Platform Prediction Studio pretrained machine learning models

Table 67: Pega Platform further aspects summary

Accessibility and Convenience of Use	
• user-friendly *App Studio* is intuitive and convenient to use • *Dev Studio* is more demanding and requires some training first • use of proprietary modeling languages	
Modes of Use	
Platform (i.e., Application Development)	*Application*
• version control system, “agile workbench”, “developer workbench” shall support collaborative development • allows shared access to user stories, comments, identified bugs, and feedback • detailed specification of different developer roles and rights in *Dev Studio*	• different application users can be assigned different GUI screens • end users are assigned a “persona” that specifies CRUD-based rights
Focus on Deployment and Scalability	
Target Location	*Support*
• cloud-based platform, accessible via a regular web browser	• possibility to define custom branches • support for Docker deployment
Focus on Artificial Intelligence	
• provision of some generic pre-trained ML models • support to integrate Google AI and SageMaker services	

5.4.3.3 Pega Platform: Conclusion

Emphasized Areas of Application Development. The Pega Platform with its four integrated workspaces (*App Studio*, *Dev Studio*, *Admin Studio*, *Prediction Studio*) is a comprehensive LCP with no single point-of-focus. The extensive catalog of rule types in *Dev Studio* supplies users with predefined structures to specify functions. Several concepts to define and manage data are included (e.g., data transforms and data pages) along two different representations of the user-defined data model (the "Visual Data Model" and "Integration Map"). Workflow types can be specified through stages and steps, for the latter various functions are pre-implemented. Diagrams for "processes" (which are a part of workflows in Pega) and more advanced configuration options can be accessed in *Dev Studio*. Collaborative development is supported through integrated communication channels where user stories, bugs, and general comments can be exchanged. Multiple types of GUIs (mobile, mashup, portal) can be developed via drag-and-drop and source code editors. A separate workspace (*Prediction Studio*) offers support for the definition of AI models and access to external AI services. Application development with the Pega Platform also faces some shortcomings. The proprietary data modeling languages do not allow to visually adjust the data models. Common modeling concepts like relationships and inheritance (which are technically supported by Pega) are not included at all. Extensive workflow types are hardly legible with the chosen representation. Diagram-based representations are only accessible in Dev Studio.

Provision of Abstractions. Implementation-related details are largely faded out. Development and configuration in *Dev Studio* is more extensive than in *App Studio*. For example, it is possible to add custom scripts, specify object-relational mappings, and define database keys. Presumably, *Dev Studio* is thus largely inaccessible to lay developers. The focus lies on supporting developers in performing repetitive tasks and providing an architecture that fosters reuse across multiple applications. The "application layer" term central for this purpose is also present in *App Studio*, where higher level abstractions from the technical particularities shall support lay developers. The platform does not offer any domain-specific abstractions to guide the definition of entity types, GUIs, and case types. Lay developers, who use *App Studio* exclusively, do not have the possibility to configure different application layers and rulesets. This leaves the definition of domain-specific abstractions and functionalities with a varying degree of specificity to professional developers that use *Dev Studio*.

Role of IT Professionals. It seems that the two application development workspaces are tailored towards the distinction between lay and professional developers. Both kinds of developers are explicitly addressed in Pega's marketing. The role of IT professionals would thus correspond to the effective use of *Dev Studio*. This encompasses, among others, the following aspects: detailed configuration that is not supported through *App Studio*, specification of further (script-based) functionalities, and clarification of technological details (e.g., database keys) to improve

efficiency and minimize integrity threats. Productivity increase as realized through complementing "application layers" pose an additional challenge and opportunity for professional developers: they need to define powerful artefacts that can be applied to a broad range of applications. More generally, *App Studio* developers can implement basic, use case-specific functionalities that meet certain requirements, while *Dev Studio* developers manage application landscapes and define artefacts suitable for reuse.

5.4.4 "Low-Code" Multi-Use Platforms for Business Application Configuration, Integration, and Development: Conclusion

All three platforms encompass some support for data modeling, workflow modeling, collaborative application development, GUI configuration, and AI services. The scope and quality of this support varies among the three vendors. While Microsoft and Pega offer several reference AI models with a user-friendly interface to specify, train, and deploy these models, Appian merely assists in the integration of external AI services. None of the three LCPs is free of shortcomings. These include the lack of basic data modeling concepts in Appian and Pega as well as the restricted modification options for reference functions in Microsoft Power Apps.

Given the general breadth of the platform's features, the choice of a platform is less concerned with the scope of feasible applications, but rather with other requirements that are tailored towards the specific use cases aimed to be implemented. All three LCPs emphasize the importance of reusing application artefacts to increase development productivity. Microsoft addresses this concern through their catalog of reference entity types, Appian relies on so-called "design objects" that exist application-independent, and Pega implements "application layers" that encompass reusable artefacts on multiple user-specified levels. While the platforms provide mechanisms to define reusable artefacts, they also lack domain-specific reference models to support developers in this regard. Appian and Pega implement hardly any domain-specific abstractions and outsource this concern to their respective marketplaces.

An assessment of the reusability of artefacts needs to consider both their respective scope and effectiveness. Therefore, this concern combines aspects from a business-oriented perspective (e.g., domain-specific concepts and requirements) and a more technology-oriented perspective (e.g., integration concerns). It is thus not surprising that the three low-code vendors in this prototypical category address lay and professional developers in their marketing. Microsoft Power Apps enables the specification of different app types with access to various editors tailored towards differently trained developers. Pega provides two integrated development environments in *App Studio* and *Dev Studio*, which are tailored towards lay and professional developers respectively. Appian's approach of implementing "design objects" fails to provide lay developers with support. Its *Quick Apps Designer* covers a scope of functionality too restricted as to be considered a reasonable alternative. Nonetheless, the considered LCPs fail to

provide convincing solutions to cope with a diverse set of differently trained and experienced developers.

6 Discussion

The subject of this study is unusual for an academic project. An investigation of available products needs to be handled with caution for reasons elucidated in the introduction. At the same time, however, a new buzzword of considerable scope implies a need for clarification – on the one hand, to treat it in a sound manner in teaching, on the other hand, to support decision-makers in practice in developing an appropriate assessment. In addition, a further aspect contributed to our interest in the topic. In recent years, a phenomenon of central importance for the future of modeling research has been discussed again and again at conferences. While the central importance of modeling for the analysis and design of complex software systems is indisputable, conceptual modeling is still not a common practice in software development processes. Instead, it is even seen by some as a cost-driver with little value only. This somewhat paradox situation leads to the question how the undisputed relevance of conceptual modeling can be demonstrated more effectively to decision makers in practice. Corresponding discussions not infrequently led to pronounced perplexity.

Against this background, our research is also aimed at finding clues on how to communicate basic ideas of conceptual modeling in such a way that they arouse the interest of decision-makers. In other words: what could academia learn from the marketing professionals of tool vendors to better communicate the case of conceptual modeling? The investigation of the tools has also stimulated interest in a fundamental research question. Various vendors do not only target novice or lay users (“citizen developers”), but also professional software developers. Accordingly, these providers are not only concerned with supporting the development of smaller applications, but also of large, enterprise-level systems. Such a claim leads to the question how to represent a tool and the artefacts it provides at different levels of abstraction.

Before we summarize our main findings (subchapter 6.1) and look at the limitations of our study (subchapter 6.2), we would like to stress one aspect that we underestimated at the beginning. With respect to the insights we gained, the effort required to develop an assessment of the tools was remarkable and has far exceeded our initial estimates. Available trainings and documentations of LCPs are sometimes strongly intertwined with marketing messages, the variety of tool vendors rely on different foundational terms, and to explore the details of an LCP – especially with a critical stance – often required to perform an extensive in-depth investigation beyond the documented set of features. While our analysis revealed characteristic commonalities of the platforms, it also showed a considerable, sometimes subtle variety of the tools.

6.1 Key Findings

The challenges addressed in the marketing of LCPs are hardly novel and approaches to deal with them have been presented in academia and business products for long (see chapter 2). Every single one of the examined LCPs existed before the advent of the low-code notion. The advertised promises seem similarly consistent over time: empowerment of business users and decrease of development time are at the center of marketing messages for most vendors.

All platforms share a common base assumption in that (business) organizations depend on software and on the ability to quickly realize and maintain software systems that satisfy changing business needs. This is such a fundamental premise that only some few vendors, like Quickbase and TrackVia, address this aspect explicitly in their marketing. Building upon this, the vendors of the platforms we investigated advertise two aspects with a varying degree of emphasis: (1) common source-code based application development is cumbersome, not efficient, and should therefore be avoided where possible. (2) Traditional application development can be supported by artefacts, mechanisms, and concepts that increase development productivity. Vendors like TrackVia address (1) almost exclusively in their marketing, while vendors like Appian emphasize (2). Yet others, e.g., Zoho or Creatio, advertise both aspects as central to low-code development. While certain features, like GUI design or the definition of data structures are characteristic for all LCPs we analyzed, the diversity of particular implementations is a clear obstacle to their comparison.

6.1.1 Low-Code Development and Source Code Configuration

One aspect that clearly illustrates this heterogeneity is a vendor's stance towards source code access and configuration. Some LCP vendors (Quickbase, TrackVia) do not provide for source code editing or the integration of external source code files. This is closely intertwined with their respective marketing messages: traditional software development in general, and coding in particular, are described as a burden that should be overcome to make companies more agile and competitive. For this purpose, source code is faded out as much as possible and configuration options typically do not go beyond simple custom arithmetic or conditional statements that resemble common spreadsheet formulas. The resemblance to spreadsheets is also explicitly advertised in low-code basic data management platforms. Whether or not this marketing image of "overcoming IT" reflects a common nerve among frustrated business users cannot be evaluated in this report. The other considered platforms (Bonita Studio, Creatio Studio, Mendix, WaveMaker, Zoho Creator, Microsoft Power Apps, Appian, Pega Platform) allow to access and edit source code to a varying degree. Most include built-in source code editors (Bonita Studio, Creatio Studio, WaveMaker, Zoho Creator, Appian, Pega Platform) so that users can define custom scripts. Some allow for the specification of functions that can be used across applications (WaveMaker, Zoho Creator, Appian, Pega Platform), which helps to avoid functional redundancy. In general, the more code editing is enabled by an LCP, the more

demanding is its use.. It is therefore not surprising that most that most vendors of platforms which allow for extensive coding address professional developers (Bonitasoft, Mendix, WaveMaker, Zoho, Microsoft, Pega Platform). Bonita Studio and WaveMaker are even marketed for use by professional developers only. It is thus clear that application development through LCPs does not address the so-called "citizen developers" exclusively nor necessarily.

6.1.2 Low-Code Development and the Provision of Abstractions

Increasing the level of abstraction from a technological point-of-view has been a central concern for computer scientists since the emergence of the discipline. This is most notably apparent in the development of programming languages. Other approaches (see chapter 2) attempt to provide powerful abstractions from implementation-level documents through the provision of conceptual models. Some form of visual model editor is apparent in every single LCP. Some platforms provide visual model editors for entity types (WaveMaker) or workflows (Bonita Studio, Creatio Studio) exclusively, whereby it should be noted that the concept of a "workflow" is not used consistently among the platforms. Most LCPs provide multiple model editors for various kinds of models (Quickbase, TrackVia, Mendix Studio Pro, Zoho Creator, Microsoft Power Apps, Appian, Pega). The modeling languages are not accessible to the user so that any adjustment or assessment of them is not possible. Some modeling languages hardly lead to the known benefits of conceptual models mentioned in chapter 2. Consider for example the rudimentary data modeling languages in Appian, Zoho Creator, and Pega Platform. Some vendors advertise that their platforms use common modeling standards, most LCPs rather use proprietary adaptations of standard modeling languages (e.g., BPMN in Creatio and Appian). The role and relevance of conceptual models for LCPs also vary. In platforms like Mendix, Creatio Studio, or Bonita Studio, conceptual models are to some extent the only accessible mechanisms for users to define database schemata, data mappings, or workflow types. Also, some platforms use conceptual models merely as an additional representation which cannot be configured by a user. This is the case for the conceptual models available on Pega Platform and the data models in Zoho Creator and Creatio Studio. In these cases, application development is supported through the provision of predefined web forms. Configuration of application artefacts through various representations is scarcely supported – although this, it might be proposed, is a central aspect to meet the requirements of different kinds of developers. Data models in WaveMaker and some types of workflow models in Zoho Creator serve as exceptions, where conceptual models are synchronized with source code and both representations can be edited by the user. In how far LCPs can be considered tools for model-based or model-driven development thus varies with each platform. Therefore, we would not regard low-code as equivalent to model-driven development as proposed by Cabot (2020), even though they share obvious commonalities.

Users are not provided with the possibility to define DSMLs on their own in any of the considered LCPs. Domain-specific support is generally scarce and not very elaborate. Most platforms offer some pre-implemented data types (e.g., currency) or entity types (e.g., customer) that correspond to general terms used in business. Beyond this, however, domain-specific support is only partially addressed through the provision of reference models. Mendix, Appian, and Pega outsource this concern to platform-specific marketplaces where domain-specific artefacts can be acquired. Creatio Studio, Zoho Creator, and Microsoft Power Apps provide a more or less extensive catalog of reference models.

6.2 Limitations

The results of our study is put into perspective by various restrictions, some of which are already mentioned in passing during the analysis of platform features (see chapter 5) or are elaborated in the depiction of the analysis method (see chapter 4).

Our analysis framework is not exhaustive and also disregards some aspects of application development in general, such as IT security concerns or economies. We do not wish to mitigate the relevance of such aspects for businesses and academia, but the investigation of such aspects – as far as it is feasible at all – does not contribute to our primary research goals. Some further aspects like performance or scalability were not accounted for because that would require the prototypical implementation of larger applications, which was clearly beyond the scope of our current study. For similar reasons, we were not able to account for additional phases of the application lifecycle, like testing, debugging, or requirements engineering.

Since our study was restricted to freely available LCPs, we were also not granted complete access to the platforms' inner-workings. For example, the underlying specification of modeling languages or data persistence mechanisms were to the most degree not accessible to us. With respect to this lack of accessibility as well as the complexity of the tools, we cannot exclude that we might have overlooked or misunderstood some features. Furthermore, the results of the study may be compromised by the fact that we were dealing with a moving target. Therefore, it is unclear for what time frame the particular results will stay valid.

7 Opportunities for Future Research

The relevance of the subject for research in business informatics is obvious. First, methods and theories that support the construction and adaptation of information systems are at the core of our discipline. Second, the continuing penetration of the world with software demands for approaches to empower people (not just the traditional "user") to understand and adapt the software that not only shapes their work environment, but more and more their entire lives. That demands for representations of software and the surroundings it is used in, which correspond to concepts people are familiar with, or that they may have to acquire to cope with "a world that is being rebuilt as code" (Widdicombe, 2014, p. 56).

The following list of research opportunities is not meant to be extensive. It rather aims at illustrating the bandwidth of inspiring research topics related to aspects of the low-code trend.

Criteria and guidelines to support the effective use of LCPs: As our analysis has shown, LCPs offer features that are suited to substantially improve software development productivity. However, this is not the case for all kinds of problems and for all kinds of organizations. Therefore, there is need to analyze preconditions of using certain kinds of LCPs, such as skills of the staff, available resources and characteristics of problem classes, and expectable economic benefits. Corresponding studies would not only address inspiring research questions but could also produce results to support decision makers in practice.

New approaches to support the development and dissemination of reference models: There is hardly any other research subject that generated as much attention and enthusiasm as reference models. The idea is indeed very appealing. Reference models are not only suited to reduce development costs and to enable a higher quality of information systems at the same time, but they are also suited to promote (cross-organizational) integration. Despite these undisputed benefits, domain-specific reference models did not become the game changers they were supposed to be. Approaches to adopt the model of open-source software (Frank & Strecker, 2007) were of limited success only. While models have not lost their attractiveness, the relevant context has partially changed. Therefore, research on obstacles of reference models and approaches to overcome them may reveal new, valuable insights.

New approaches to enable process automation: In recent years, approaches to support non-programmers with the automation of processes has received growing awareness. This is especially the case in areas such as "smart home" (M. Clark, Dutta, & Newman, 2016; Stefanidi, Korozi, Leonidis, & Antona, 2018) or, more general, IoT (e.g., Valsamakis & Savidis, 2020). Corresponding research opportunities comprise the development of accessible abstractions of "smart" environments and of concepts that facilitate the construction of simple processes.

Mitigating the conflict between range and productivity of reuse: while reuse is of pivotal relevance for promoting the productivity of software development, it is confronted with a fundamental

design conflict. The more specific a reusable artefact is, the better is its contribution to reuse, but the lower is the range of cases it covers, and, hence, the possible economies of scale (Frank, 2014). Various approaches exist to relax it, such as generalization/specialization or multi-level language architectures. Nevertheless, there is still need for further research. One promising approach is the integration of DSML with reference models that allows combining the benefits of generalization with those of classification (Kinderen & Kaczmarek-Heß, 2019). Corresponding research would suggest cross-disciplinary collaboration with representatives of programming languages and software engineering communities.

Beyond spreadsheets: spreadsheet programs proved to be an effective approach to support non-programmers with the creation of certain kinds of (small) information systems. However, since they are based on a generic model, their contribution to productivity as well as their support of data integrity are limited. Applying the idea of generic spreadsheets to domain-specific models of data and related processes would enable the creation of more productive tools that could feature a high level of data integrity, too (Carroll, 2019; Sternberg, 2020).

Cognitive fit of representations: Even though low-code platforms in part address both, professional software developers and “citizen developers”, it remains unclear what kind of user models they employ. However, for a representation of software to fit cognitive capabilities and personal working styles, corresponding models are required. In addition, the suitability of a representation is likely to depend on peculiarities of the task and contextual factors such as availability of resources. The investigation of this topic opens a wide range of research questions involving fields such as cognitive psychology or modeling/programming language design.

Prospects and limitations of machine learning: Some of the tools we analyzed employ machine learning algorithms already, but provide a modest level of support, such as "auto-complete" functions with model editors that are based on the inductive analysis of previous decisions. Proponents of machine learning research predict a clearly more ambitious vision: the construction of programs by machines. There are indeed impressive examples of software created to a large extent through machine learning, such as speech or image recognition. Nevertheless, that is hardly sufficient to conclude that conceptual modeling will become obsolete, as Domingos suggests: "In industry, there’s no sign that knowledge engineering will ever be able to compete with machine learning outside of a few niche areas."

While such a prediction may be seen as visionary by some, and as offensive by others, it concerns fundamental and extremely challenging research questions, such as "What is the difference between classification and conceptualization?", "What is the nature of application software?", or "Is it possible to inductively create software that supports future, not yet existing business models, from examples created in the past?" (Domingos, 2017, p. 36).

Effects on organizational behavior: Approaches to foster end-user computing are suited to blur traditional boundaries between developers and users. Furthermore, they may contribute to users develop a more elaborate appreciation of IT. Both aspects are likely to have an effect on decision processes, patterns of collaboration, and on organizational behavior in general (Amoroso, 1988), which calls for empirical research aimed at identifying and explaining such patterns.

In addition to research opportunities directly related to the objectives targeted by low-code platforms, the emergence of the low-code trend is yet another reason to investigate the role and impact of so-called market research firms. Among others, that could comprise the following questions:

- What is their business model?
- What is their role in creating new trends - and fads?
- What methods do they apply to conduct their studies?
- How accurate were the predictions they made in the past?
- What role do their reports play for product development and for marketing activities?
- What impact do they have on managerial decision making?

8 Conclusions

The results of our study create an ambivalent picture. On the one hand, we did not find evidence for the assumption that LCPs incorporate any specific innovation that would come close to the state of the art in research or would even go beyond it. On the other hand, the "low-code" trend creates an opportunity for research on conceptual modeling since it should raise the awareness for the importance of representations other than code to developing and maintaining software systems. In this sense, the trend is suited to inspire numerous opportunities for research and its dissemination. Nevertheless, we would caution against adopting the simplistic messages and the terminology, which is inaccurate and misleading in parts.

The limitations of our study may have compromised the scientific rigor, we feel committed to. We are therefore grateful for any information that reveals misconceptions or errors in our analysis.

1. References

Agam Shah (2020). Emptying Offices Prompt Adoption of Low-Code to Build Work Apps. *The Wall Street Journal*, May 15th.

Ahmad, Y., Antoniu, T., Goldwater, S., & Krishnamurthi, S. (2003). A type system for statically detecting spreadsheet errors. In *Proceedings / 18th IEEE International Conference on Automated Software Engineering: Montreal, Quebec, Canada, October 6 to 10, 2003* (pp. 174–183). Los Alamitos, Calif.: IEEE Computer Society. https://doi.org/10.1109/ASE.2003.1240305

Amoroso, D. L. (1988). Organizational issues of end-user computing. *SIGMIS Database*, *19*(3-4), 49–58. https://doi.org/10.1145/65766.65773

Becker, J., & Delfmann, P. (Eds.) (2007). *Reference modeling: Efficient information systems design through reuse of information models*. Heidelberg: Physica Verlag.

Becker, J., Delfmann, P., & Knackstedt, R. (2007). Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In J. Becker (Ed.), *Reference modeling: Efficient information systems design through reuse of information models : [9th conference on reference modeling (RefMod 2006) taken place on the Multiconference on Information Systems (Multi-Konferenz Wirtschaftsinformatik, MKWI 2006) the 20th of February 2006* (pp. 27–58). Heidelberg: Physica-Verl. https://doi.org/10.1007/978-3-7908-1966-3_2

Bock, A. C., & Frank, U. (2021a). In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms. In *Proceedings of the 24th ACM/IEEE International Conference on Modell Driven Engineering Languages and Systems: Companion Proceedings*. IEEE.

Bock, A. C., & Frank, U. (2021b). Low-Code Platform. *Business & Information Systems Engineering*, *63*(6).

Bohrer, K. A. (1998). Architecture of the San Francisco frameworks. *IBM Systems Journal*, *37*(2), 156–169. https://doi.org/10.1147/sj.372.0156

Brambilla, M., Cabot, J., Wimmer, M., & Baresi, L. (2017). *Model-Driven Software Engineering in Practice: Second Edition*. *Synthesis Lectures on Software Engineering*. San Rafael: Morgan & Claypool Publishers. Retrieved from https://ebookcentral.proquest.com/lib/gbv/detail.action?docID=4837000

Cabot, J. (2020). Positioning of the low-code movement within the field of model-driven engineering. In *Proceedings of the 23rd ACM/IEEE International Conference on Modell Driven Engineering Languages and Systems: Companion Proceedings* (pp. 535–538). IEEE. https://doi.org/10.1145/3417990.3420210

Carroll, J. (2019). *Beyond spreadsheets with R: A beginner's guide to R and RStudio*. Shelter Island, NY: Manning.

Chang Young-Hyun, & Ko Chang-Bae (2017). A Study on the Design of Low-Code and No Code Platform for Mobile Application Development. *International Journal of Advanced Smart Convergence*, *6*(4), 50–55. https://doi.org/10.7236/IJASC.2017.6.4.7

Clark, M., Dutta, P., & Newman, M. W. (2016). Towards a natural language programming interface for smart homes. In P. Lukowicz, A. Krüger, A. Bulling, Y.-K. Lim, & S. N. Patel (Eds.), *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct : September 12- 16, 2016, Heidelberg, Germany* (pp. 49–52). New York, NY: ACM. https://doi.org/10.1145/2968219.2971443

Clark, T., Sammut, P., & Willans, J. (2008). *Applied Metamodelling: A Foundation for Language Driven Development* (2nd ed.). Ceteva. Retrieved from http://www.eis.mdx.ac.uk/staffpages/tonyclark/Papers/Applied%20Metamodelling%20%28Second%20Edition%29.pdf

Codenie, W., Hondt, K. de, Steyaert, P., & Vercammen, A. (1997). From custom applications to domain-specific frameworks. *Commun. ACM*, *40*(10), 70–77. https://doi.org/10.1145/262793.262807

Costagliola, G., Deufemia, V., & Polese, G. (2004). A framework for modeling and implementing visual notations with applications to software engineering. *ACM Transactions of Software Engineering Methodologies*, *13*(4), 431–487. https://doi.org/10.1145/1040291.1040293

Domingos, P. (2017). *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. London: Penguin Books Ltd.

Dou, W., Cheung, S.-C., & Wei, J. (2014). Is spreadsheet ambiguity harmful? Detecting and repairing spreadsheet smells due to ambiguous computation. In P. Jalote (Ed.), *Proceedings of the 36th International Conference on Software Engineering: Hyderabad, India, May 31 - June 07, 2014* (pp. 848–858). New York, NY: Assoc. for Computing Machinery. https://doi.org/10.1145/2568225.2568316

Fayad, M. E., & Johnson, R. E. (Eds.) (2000). *Domain-specific application frameworks: Frameworks experience by industry*. New York, NY: Wiley. Retrieved from http://www.loc.gov/catdir/bios/wiley042/99026920.html

Fettke, P., & Loos, P. (Eds.) (2007). *Reference Modeling for Business Systems Analysis*. Hershey: Idea Group.

France, R. B., & Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. In L. C. Briand & A. L. Wolf (Eds.), *Workshop on the Future of Software Engineering (FOSE '07)* (pp. 37–54). IEEE CS Press. Retrieved from http://sse-tubs.de/publications/FR_MDDofComplexSoftware_ICSE_07.pdf

Frank, U. (2007). Evaluation of Reference Models. In P. Fettke & P. Loos (Eds.), *Reference Modeling for Business Systems Analysis* (pp. 118–140). Hershey: Idea Group.

Frank, U. (2011a). *MEMO Organisation Modelling Language (1): Focus on Organisational Structure* (ICB Research Report No. 48).

Frank, U. (2011b). *Multi-Perspective Enterprise Modelling: Background and Terminological Foundation* (ICB Research Report No. 46). Retrieved from ICB University of Duisburg-Essen, Campus Essen website: http://www.icb.uni-due.de/filead-min/ICB/research/research_reports/ICB-Report-No46.pdf

Frank, U. (2012). *Specialisation in Business Process Modelling: Motivation, Approaches and Limitations* (ICB Research Report No. 51). Essen. Retrieved from ICB University of Duisburg-Essen, Campus Essen website: http://www.icb.uni-due.de/filead-min/ICB/research/research_reports/ICB-Report_No51.pdf

Frank, U. (2014). Multilevel Modeling: Toward a New Paradigm of Conceptual Modeling and Information Systems Design. *Business and Information Systems Engineering*, *6*(6), 319–337.

Frank, U. (2016). Designing Models and Systems to Support IT Management: A Case for Multilevel Modeling. In *Proceedings of MULTI 2016* (pp. 3–24). Retrieved from http://www.wi-inf.uni-due.de/FGFrank/documents/Konferenzbeitraege/ML-ITML-Multi2016.pdf

Frank, U., & Bock, A. (2020). Conjoint Analysis and Design of Business and IT: The Case for Multi-Perspective Enterprise Modeling. In V. Kulkarni, S. Reddy, T. Clark, & B. Barn (Eds.), *Advanced Digital Architectures for Model-Driven Adaptive Enterprises* (pp. 15–45). IGI Global.

Frank, U., & Strecker, S. (2007). Open Reference Models – Community-driven Collaboration to Promote Development and Dissemination of Reference Models. *Enterprise Modelling and Information Systems Architectures*, *2*(2), 32–41.

Garriga, M., & Flores, A. (2019). Standards-driven metamodel to increase retrievability of heterogeneous services. In C.-C. Hung & G. A. Papadopoulos (Eds.), *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (pp. 2507–2514). New York, NY, USA: ACM. https://doi.org/10.1145/3297280.3297527

Gulden, J., & Frank, U. (2010). MEMOCenterNG – A full-featured modeling environment for organisation modeling and model-driven software development. In *Proceedings of the 2nd International Workshop on Future Trends of Model-Driven Development (FTMDD 2010)*, Funchal, Portugal.

Hermans, F., Pinzger, M., & van Deursen, A. (2011). Supporting professional spreadsheet users by generating leveled dataflow diagrams. In R. N. Taylor, H. Gall, & N. Medvidović (Eds.), *33rd International Conference on Software Engineering (ICSE), 2011: 21 - 28 May 2011, Waikiki, Honolulu, HI, USA* (pp. 451–460). Piscataway, NJ: IEEE. https://doi.org/10.1145/1985793.1985855

Ihirwe, F., Di Ruscio, D., Mazzini, S., Pierini, P., & Pierantonio, A. (2020). Low-code engineering for internet of things. In E. Guerra & L. Iovino (Chairs), *MODELS '20: ACM/IEEE 23rd*

International Conference on Model Driven Engineering Languages and Systems, Virtual Event Canada.

Ingalls, D., Wallace, S., Chow, Y.-Y., Ludolph, F., & Doyle, K. (1988). Fabrik: A visual programming environment. In N. Meyrowitz (Ed.), *Conference proceedings on Object-oriented programming systems, languages and applications* (pp. 176–190). New York, NY: ACM. https://doi.org/10.1145/62083.62100

Karl, H., Kundisch, D., Meyer auf der Heide, Friedhelm, & Wehrheim, H. (2020). A Case for a New IT Ecosystem: On-The-Fly Computing. *Business & Information Systems Engineering*, *62*(6), 467–481. https://doi.org/10.1007/s12599-019-00627-x

Kärnä, J., Tolvanen, J.-P., & Kelly, S. (2009). Evaluating the use of domain-specific modeling in practice. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling*.

Kelly, S., & Tolvanen, J.-P. (2008). *Domain-specific modeling: Enabling full code generation*. Hoboken, N.J: Wiley-Interscience; IEEE Computer Society.

Kinderen, S. D., & Kaczmarek-Heß, M. (2019). Multi-level Modeling as a Language Architecture for Reference Models: On the Example of the Smart Grid Domain. In IEEE (Ed.), *IEEE 21st Conference on Business Informatics (CBI)* (pp. 174–183). Moscow, Russia.

Lieberman, H., Paternò, F., Klann, M., & Wulf, V. (2006). End-User Development: An Emerging Paradigm. In H. Lieberman, F. Paternò, & V. Wulf (Eds.), *Human-Computer Interaction Series: Vol. 9. End User Development* (Vol. 9, pp. 1–8). Dordrecht: Springer. https://doi.org/10.1007/1-4020-5386-X_1

Massoni, T., Gheyi, R., & Borba, P. (2011). Synchronizing Model and Program Refactoring. In J. Davies, L. Silva, & A. Simao (Eds.), *Lecture Notes in Computer Science: Vol. 6527. Formal methods: foundations and applications: 13th Brazilian Symposium on Formal Methods, SBMF 2010, Natal, Brazil, November 8-11, 2010 ; revised selected papers* (Vol. 6527, pp. 96–111). Heidelberg: Springer. https://doi.org/10.1007/978-3-642-19829-8_7

Mendes, J., Cunha, J., Duarte, F., Engels, G., Saraiva, J., & Sauer, S. (2017). Towards systematic spreadsheet construction processes. In *2017 IEEE/ACM 39th International Conference on Software Engineering companion: Icse-C 2017 : 20-28 May 2017, Buenos Aires, Argentina : Proceedings* (pp. 356–358). Piscataway, NJ: IEEE. https://doi.org/10.1109/ICSE-C.2017.141

Milani, F., Dumas, M., Ahmed, N., & Matulevičius, R. (2016). Modelling families of business process variants: A decomposition driven method. *Information Systems*, *56*, 55–72. https://doi.org/10.1016/j.is.2015.09.003

Moulin, C., & Sbodio, M. (2005). Using Ontological Concepts for Web Service Composition. In A. Skowron (Ed.), *Proceedings / The 2005 IEEE/WIC/ACM International Conference on Web Intelligence: September 19 - 22, 2005, Compiègne University of Technology, France* (pp. 487–490). Los Alamitos, Calif.: IEEE Computer Society. https://doi.org/10.1109/WI.2005.156

Nardi, B. A. (1995). *A small matter of programming: Perspectives on end user computing* (2. printing). Cambridge, Mass.: MIT Press.

Naur, P., & Randell, B. (Eds.) (1969). *Software engineering: Report on a conference sponsered by the NATO Science Committee Garmisch, 7th-11th October, 1968*. Brussels: NATO, Scientific Affairs Division.

Paige, R. F., Hartman, A., & Rensink, A. (Eds.) (2009). *Lecture Notes in Computer Science: Vol. 5562. Model Driven Architecture - Foundations and Applications: 5th European Conference, ECMDA-FA 2009, Enschede, The Netherlands, June 23-26, 2009. Proceedings*. Berlin, Heidelberg: Springer. Retrieved from http://dx.doi.org/10.1007/978-3-642-02674-4

Prieto-Díaz, R. (1991). Implementing faceted classification for software reuse. *Commun. ACM*, *34*(5), 88–97. https://doi.org/10.1145/103167.103176

Razavi, A., Kontogiannis, K., Brealey, C., & Nigul, L. (2009), usuIncremental model synchronization in model driven development environments. In P. Martin, A. W. Kark, & D. Stewart (Eds.), *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research - CASCON '09* (p. 216). New York, New York, USA: ACM Press. https://doi.org/10.1145/1723028.1723053

Sanchis, R., García-Perales, Ó., Fraile, F., & Poler, R. (2020). Low-Code as Enabler of Digital Transformation in Manufacturing Industry. *Applied Sciences*, *10*(1), 12. https://doi.org/10.3390/app10010012

Schneider, J.-G. (1999). *Components, Scripts, and Glue: A conceptual framework for software composition*. Dissertation, University of Bern.

Smirnov, S., Weidlich, M., & Mendling, J. (2007). Business Process Model Abstraction Based on Behavioral Profiles. In B. J. Krämer, K.-J. Lin, & P. Narasimhan (Eds.), *Lecture Notes in Computer Science: Vol. 4749. Service-oriented computing - ICSOC 2007: Fifth international conference, Vienna, Austria, September 17 - 20, 2007 ; proceedings* (Vol. 4749, pp. 1–16). Berlin: Springer. https://doi.org/10.1007/978-3-642-17358-5_1

St. Amant, R., Lieberman, H., Potter, R., & Zettlemoyer, L. (2000). Programming by example: visual generalization in programming by example. *Commun. ACM*, *43*(3), 107–114. https://doi.org/10.1145/330534.330549

Stefanidi, E., Korozi, M., Leonidis, A., & Antona, M. (2018). Programming Intelligent Environments in Natural Language. In *ICPS, Petra 2018: The 11th ACM International Conference on Pervasive Technologies Related to Assistive Environments : June 26-29, 2018, Corfu, Greece : Conference proceedings* (pp. 50–57). New York, NY, USA: ACM. https://doi.org/10.1145/3197768.3197776

Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2009). *EMF: Eclipse Modeling Framework* (2nd ed.). Upper Saddle River: Addison-Wesley.

Sternberg, J. (2020). Beyond Spreadsheets. *Forbes*. (March 11th). Retrieved from https://www.forbes.com/sites/googlecloud/2020/03/11/beyond-spreadsheets/?sh=d10ecb076c7f

Tilsner, M., Fiech, A., Zhan, G., & Specht, T. (2011). Patterns for service composition. In B. C. Desai (Ed.), *Proceedings of The Fourth International C* Conference on Computer Science and Software Engineering* (pp. 133–137). New York, NY: ACM. https://doi.org/10.1145/1992896.1992913

Valsamakis, Y., & Savidis, A. (2020). Smart Automations for Everybody: When IoT Meets Visual Programming. In P. Davidsson (Ed.), *ACM Digital Library, 10th International Conference on the Internet of Things Companion* (pp. 1–4). New York,NY,United States: Association for Computing Machinery. https://doi.org/10.1145/3423423.3423470

Vincent, P., Natis, Y., & et al. (2020). *Magic Quadrant for Enterprise Low-Code Application Platforms. Gartner Report.*

Virgilio, R. de, & Bianchini, D. (2010). A metamodel approach to flexible semantic web service discovery. In J. Huang (Ed.), *Proceedings of the 19th ACM international conference on Information and knowledge management* (p. 1309). New York, NY: ACM. https://doi.org/10.1145/1871437.1871608

Völter, M. (2013). *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. dslbooks.org.

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

Weber, I., Paik, H.-Y., & Benatallah, B. (2013). Form-Based Web Service Composition for Domain Experts. *ACM Transactions on the Web*, *8*(1), 1–40. https://doi.org/10.1145/2542168

Widdicombe, L. (2014). The Programmer's Price. *The New Yorker*. (Nov. 24), 54–64.

Wyatt, F. J. (2018). *New Development Platforms Emerge For Customer-Facing Applications*. Retrieved from https://medium.com/business-process-management-software-comparisons/new-development-platforms-emerge-for-customer-facing-applications-f5b1f7831a3e

Zloof, M. M. (1975). Query by example. In Unknown (Ed.), *Proceedings of the May 19-22, 1975, national computer conference and exposition on - AFIPS '75* (p. 431). New York, New York, USA: ACM Press. https://doi.org/10.1145/1499949.1500034

Previously published ICB - Research Reports

2021

No 69 (June 2021)
Schauer, Carola: "Wirtschaftsinformatik Studiengänge an Universitäten in Deutschland – Analyse der Studienanfängerzahlen und Frauenanteile im Vergleich zur Informatik und zu Fachhochschulen."

2020

No 68 (December 2020)
Schauer, Carola: "Warum entscheiden sich Studienanfänger für Wirtschaftsinformatik? – Ergebnisse einer Umfrage unter Bachelorstudierenden im ersten Fachsemester Wirtschaftsinformatik an der UDE (Nov. 2018 und Nov. 2019)."

No 67 (November 2020)
Frank, Ulrich; Bock, Alexander: "Organisationsforschung und Wirtschaftsinformatik: Zeit für eine Annäherung?"

2018

No 66 (December 2019)
Frank, Ulrich: "The Flexible Multi-Level Modelling and Execution Language (FMMLx, Version 2.0: Analysis of Requirements and Technical Terminology."

2015

No 65 (August 2015)
Schauer, Carola; Schauer, Hanno: "IT- und Medienbildung an Schulen. Ergebnisse einer empirischen Studie an einem rheinland-pfalzischen Gymnasium."

No 64 (January 2015)
Föcker, Felix; Houdek, Frank; Daun, Marian; Weyer, Thorsten: "Model-Based Engineering of an Automotive Adaptive Exterior Lighting System – Realistic Example Specifications of

Behavioral Requirements and Functional Design."

No 63 (January 2015)
Schauer, Carola; Schauer, Hanno: "IT an allgemeinbildenden Schulen: Bildungsgegenstand und -infrastruktur – Auswertung internationaler empirischer Studien und Literaturanalyse."

2014

No 62 (October 2014)
Köninger, Stephan; Hes, Michael: "Ein Software-Werkzeug zur multiperspektivischen Bewertung innovativer Produkte, Projekte und Dienstleistungen: Realisierung im Projekt Hospital Engineering."

No 61 (August 2014)
Schauer, Carola; Frank, Ulrich: "Wirtschaftsinformatik an Schulen – Status und Desiderata mit Fokus auf Nordrhein-Westfalen."

No 60 (May 2014)

Hes, Michael: "Multiperspektivische Dokumentation und Informationsbedarfsanalyse kardiologischer Prozesse sowie Konzeptualisierung ausgewählter medizinischer Ressourcentypen im Projekt Hospital Engineering"

No 59 (May 2014)

Goedicke, Michael; Kurt-Karaoglu, Filiz; Schwinning, Nils; Schypula, Melanie; Striewe, Michael: "Zweiter Jahresbericht zum Projekt 'Bildungsgerechtigkeit im Fokus' (Teilprojekt 1.2 – 'Blended Learning') an der Fakultät für Wirtschaftswissenschaften"

No 58 (March 2014)

Breitschwerdt, Rüdiger; Hes, Michael: "Konzeption eines Bezugsrahmens zur Analyse und Entwicklung von Geschäftsmodellen mobiler Gesundheitsdienstleistungen – Langfassung"

No 57 (March 2014)

Hes, Michael; Schlieter, Hannes (Hrsg.): "Modellierung im Gesundheitswesen – Tagungsband des Workshops im Rahmen der "Modellierung 2014""

2013

No 56 (July 2013)

Svensson, Richard Berntsson; Berry, Daniel M.; Daneva, Maya; Doerr, Joerg; Espana, Sergio; Herrmann, Andrea; Herzwurm, Georg; Hoffmann, Anne; Pena, Raul Mazo; Opdahl, Andreas L.; Pastor, Oscar; Pietsch, Wolfram; Salinesi, Camille; Schneider, Kurt; Seyff, Norbert; van de Weerd, Inge; Wieringa, Roel; Wnuk, Krzysztof (Eds.): "19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2013). Proceedings of the REFSQ 2013 Workshops CreaRE, IWSPM, and RePriCo, the REFSQ 2013 Empirical Track (Empirical Live Experiment and Empirical Research Fair), the REFSQ 2013 Doctoral Symposium, and the REFSQ 2013 Poster Session""

No 55 (May 2013)

Daun, Marian; Focke, Markus; Holtmann, Jörg; Tenbergen, Bastian "Goal-Scenario-Oriented Requirements Engineering for Functional Decomposition with Bidirectional Transformation to Controlled Natural Language. Case Study "Body Control Module""

No 54 (March 2013)

Fischotter, Melanie; Goedicke, Michael; Kurt-Karaoglu, Filiz; Schwinning, Nils; Striewe, Michael "Erster Jahresbericht zum Projekt "Bildungsgerechtigkeit im Fokus" (Teilprojekt 1.2 – "Blended Learning") an der Fakultät für Wirtschaftswissenschaften"

2012

No 53 (December 2012)

Frank, Ulrich: "Thoughts on Classification / Instantiation and Generalisation / Specialisation"

No 52 (July 2012)

Berntsson-Svensson, Richard; Berry, Daniel; Daneva, Maya; Dörr, Jörg; Fricker, Samuel A; Herrmann, Andrea; Herzwurm, Georg; Kauppinen, Marjo; Madhavji, Nazim H; Mahaux, Martin; Paech, Barbara; Penzenstadler, Birgit; Pietsch, Wolfram; Salinesi, Camille; Schneider, Kurt; Seyff, Norbert; van de Weerd, Inge (Eds.): "18th International Working Conference on Requirements Engineering – Foundation

for Software Quality. Proceedings of the Workshops RE4SuSy, REEW, CreaRE, RePriCo, IWSPM and the Conference Related Empirical Study, Empirical Fair and Doctoral Symposium"

No 51 (May 2012)
Frank, Ulrich: "Specialisation in Business Process Modelling – Motivation, Approaches and Limitations"

No 50 (March 2012)
Adelsberger, Heimo; Drechsler, Andreas; Herzig, Eric; Michaelis, Alexander; Schulz, Philipp ; Schütz, Stefan; Ulrich, Udo: "Qualitative und quantitative Analyse von SOA-Studien – Eine Metastudie zu serviceorientierten Architekturen"

2011

No 47 (December 2011)
Frank, Ulrich: "MEMO Organisation Modelling Language (OrgML): Requirements and Core Diagram Types"

No 46 (December 2011)
Frank, Ulrich: "Multi-Perspective Enterprise Modelling: Background and Terminological Foundation"

No 45 (November 2011)
Frank, Ulrich; Strecker, Stefan; Heise, David; Kattenstroth, Heiko; Schauer, Carola: "Leitfaden zur Erstellung wissenschaftlicher Arbeiten in der Wirtschaftsinformatik"

No 44 (September 2010)
Berenbach, Brian; Daneva, Maya; Dörr, Jörg; Frickler, Samuel; Gervasi, Vincenzo; Glinz, Martin; Herrmann, Andrea; Krams, Benedikt; Madhavji, Nazim H.; Paech, Barbara; Schockert, Sixten; Seyff, Norbert (Eds): "17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2011). Proceedings of the REFSQ 2011 Workshops REEW, EPICAL and RePriCo, the REFSQ 2011 Empirical Track (Empirical Live Experiment and Empirical Research Fair), and the REFSQ 2011 Doctoral Symposium"

No 43 (February 2011)
Frank, Ulrich: "The MEMO Meta Modelling Language (MML) and Language Architecture – 2nd Edition"

2010

No 42 (December 2010)
Frank, Ulrich: "Outline of a Method for Designing Domain-Specific Modelling Languages"

No 41 (December 2010)
Adelsberger, Heimo; Drechsler, Andreas (Eds): "Ausgewählte Aspekte des Cloud-Computing aus einer IT-Management-Perspektive – Cloud Governance, Cloud Security und Einsatz von Cloud Computing in jungen Unternehmen"

No 40 (October 2010)
Bürsner, Simone; Dörr, Jörg; Gehlert, Andreas; Herrmann, Andrea; Herzwurm, Georg; Janzen, Dirk; Merten, Thorsten; Pietsch, Wolfram; Schmid, Klaus; Schneider, Kurt; Thurimella, Anil Kumar (Eds):

"16th International Working Conference on Requirements Engineering: Foundation for Software Quality. Proceedings oft he Workshops CreaRE, PLREQ, RePriCo and RESC"

No 39 (May 2010)
Strecker, Stefan; Heise, David; Frank, Ulrich: "Entwurf einer Mentoring-Konzeption für den Studiengang M.Sc. Wirtschaftsinformatik an der Fakultät für Wirtschaftswissenschaften der Universität Duisburg-Essen"

No 38 (February 2010)
Schauer, Carola: "Wie praxisorientiert ist die Wirtschaftsinformatik? Einschätzungen von CIOs und WI-Professoren"

No 37 (January 2010)
Benavides, David; Batory, Don; Grunbacher, Paul (Eds.): "Fourth International Workshop on Variability Modelling of Software-intensive Systems"

2009

No 36 (December 2009)
Strecker, Stefan: "Ein Kommentar zur Diskussion um Begriff und Verständnis der IT-Governance - Anregungen zu einer kritischen Reflexion"

No 35 (August 2009)
Rüngeler, Irene; Tüxen, Michael; Rathgeb, Erwin P.:"Considerations on Handling Link Errors in STCP"

No 34 (June 2009)
Karastoyanova, Dimka; Kazhamiakan, Raman; Metzger, Andreas; Pistore, Marco (Eds.): "Workshop on Service Monitoring, Adaption and Beyond"

No 33 (May 2009)
Adelsberger,Heimo; Drechsler , Andreas; Bruckmann, Tobias; Kalvelage, Peter; Kinne, Sophia; Pellinger, Jan; Rosenberger, Marcel; Trepper, Tobias: „Einsatz von Social Software in Unternehmen – Studie über Umfang und Zweck der Nutzung"

No 32 (April 2009)
Barth, Manfred; Gadatsch, Andreas; Kütz, Martin; Rüding, Otto; Schauer, Hanno; Strecker, Stefan: „Leitbild IT-Controller/-in – Beitrag der Fachgruppe IT-Controlling der Gesellschaft für Informatik e. V."

No 31 (April 2009)
Frank, Ulrich; Strecker, Stefan: "Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems – Requirements, Conceptual Foundation and Design Options"

No 30 (February 2009)
Schauer, Hanno; Wolff, Frank: „Kriterien guter Wissensarbeit – Ein Vorschlag aus dem Blickwinkel der Wissenschaftstheorie (Langfassung)"

No 29 (January 2009)
Benavides, David; Metzger, Andreas; Eisenecker, Ulrich (Eds.): "Third International Workshop on Variability Modelling of Software-intensive Systems"

2008

No 28 (December 2008)
Goedicke, Michael; Striewe, Michael; Balz, Moritz: „Computer Aided Assessments and Programming Exercises with JACK“

No 27 (December 2008)
Schauer, Carola: “Größe und Ausrichtung der Disziplin Wirtschaftsinformatik an Universitäten im deutschsprachigen Raum - Aktueller Status und Entwicklung seit 1992”

No 26 (September 2008)
Milen, Tilev; Bruno Müller-Clostermann: “ CapSys: A Tool for Macroscopic Capacity Planning”

No 25 (August 2008)
Eicker, Stefan; Spies, Thorsten; Tschersich, Markus: “Einsatz von Multi-Touch beim Softwaredesign am Beispiel der CRC Card-Methode”

No 24 (August 2008)
Frank, Ulrich: “The MEMO Meta Modelling Language (MML) and Language Architecture – Revised Version”

No 23 (January 2008)
Sprenger, Jonas; Jung, Jürgen: “Enterprise Modelling in the Context of Manufacturing – Outline of an Approach Supporting Production Planning”

No 22 (January 2008)
Heymans, Patrick; Kang, Kyo-Chul; Metzger, Andreas, Pohl, Klaus (Eds.): “Second International Workshop on Variability Modelling of Software-intensive Systems”

2007

No 21 (September 2007)
Eicker, Stefan; Annett Nagel; Peter M. Schuler: “Flexibilität im Geschäftsprozess-management-Kreislauf”

No 20 (August 2007)
Blau, Holger; Eicker, Stefan; Spies, Thorsten: “Reifegradüberwachung von Software”

No 19 (June 2007)
Schauer, Carola: “Relevance and Success of IS Teaching and Research: An Analysis of the ‚Relevance Debate’

No 18 (May 2007)
Schauer, Carola: “Rekonstruktion der historischen Entwicklung der Wirtschaftsinformatik: Schritte der Institutionalisierung, Diskussion zum Status, Rahmenempfehlungen für die Lehre”

No 17 (May 2007)
Schauer, Carola; Schmeing, Tobias: “Development of IS Teaching in North-America: An Analysis of Model Curricula”

No 16 (May 2007)
Müller-Clostermann, Bruno; Tilev, Milen: “Using G/G/m-Models for Multi-Server and Mainframe Capacity Planning”

No 15 (April 2007)
Heise, David; Schauer, Carola; Strecker, Stefan: "Informationsquellen für IT-Professionals – Analyse und Bewertung der Fachpresse aus Sicht der Wirtschaftsinformatik"

No 14 (March 2007)
Eicker, Stefan; Hegmanns, Christian; Malich, Stefan: "Auswahl von Bewertungsmethoden für Softwarearchitekturen"

No 13 (February 2007)
Eicker, Stefan; Spies, Thorsten; Kahl, Christian: "Softwarevisualisierung im Kontext serviceorientierter Architekturen"

No 12 (February 2007)
Brenner, Freimut: "Cumulative Measures of Absorbing Joint Markov Chains and an Application to Markovian Process Algebras"

No 11 (February 2007)
Kirchner, Lutz: "Entwurf einer Modellierungssprache zur Unterstützung der Aufgaben des IT-Managements – Grundlagen, Anforderungen und Metamodell"

No 10 (February 2007)
Schauer, Carola; Strecker, Stefan: "Vergleichende Literaturstudie aktueller einführender Lehrbücher der Wirtschaftsinformatik: Bezugsrahmen und Auswertung"

No 9 (February 2007)
Strecker, Stefan; Kuckertz, Andreas; Pawlowski, Jan M.: "Überlegungen zur Qualifizierung des wissenschaftlichen Nachwuchses: Ein Diskussionsbeitrag zur (kumulativen) Habilitation"

No 8 (February 2007)
Frank, Ulrich; Strecker, Stefan; Koch, Stefan: "Open Model - Ein Vorschlag für ein Forschungsprogramm der Wirtschaftsinformatik (Langfassung)"

2006

No 7 (December 2006)
Frank, Ulrich: "Towards a Pluralistic Conception of Research Methods in Information Systems Research"

No 6 (April 2006)
Frank, Ulrich: "Evaluation von Forschung und Lehre an Universitäten – Ein Diskussionsbeitrag"

No 5 (April 2006)
Jung, Jürgen: "Supply Chains in the Context of Resource Modelling"

No 4 (February 2006)
Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part III – Results Wirtschaftsinformatik Discipline"

2005

No 3 (December 2005)

Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part II – Results Information Systems Discipline"

No 2 (December 2005)

Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part I – Research Objectives and Method"

No 1 (August 2005)

Lange, Carola: „Ein Bezugsrahmen zur Beschreibung von Forschungsgegenständen und -methoden in Wirtschaftsinformatik und Information Systems"

Research Group	Core Research Topics
Prof. Dr. F. Ahlemann Information Systems and Strategic Management	Strategic planning of IS, Enterprise Architecture Management, IT Vendor Management, Project Portfolio Management, IT Governance, Strategic IT Benchmarking
Prof. Dr. F. Beck Visualization Research Group	Information visualization, software visualization, visual analytics
Prof. Dr. T. Brinda Didactics of Informatics	Competence modelling and educational standards in Informatics, Students' conceptions in Informatics, Education in the digital world, Vocational education in Informatics
Prof. Dr. P. Chamoni MIS and Management Science / Operations Research	Information Systems and Operations Research, Business Intelligence, Data Warehousing
Prof. Dr.-Ing. L. Davi Research in Secure Software Systems	Software Security, Security of Smart Contracts, Trusted Computing, Hardware-assisted Security
Prof. Dr. K. Echtle Dependability of Computing Systems	Dependability of Computing Systems
Prof. Dr. S. Eicker Information Systems and Software Engineering	Process Models, Software-Architectures
Prof. Dr. U. Frank Information Systems and Enterprise Modelling	Enterprise Modelling, Enterprise Application Integration, IT Management, Knowledge Management
Prof. Dr. M. Goedicke Specification of Software Systems	Distributed Systems, Software Components, CSCW
Prof. Dr. V. Gruhn Software Engineering	Design of Software Processes, Software Architecture, Usability, Mobile Applications, Component-based and Generative Software Development
Prof. Dr. T. Kollmann E-Business and E-Entrepreneurship	E-Business and Information Management, E-Entrepreneurship/E-Venture, Virtual Marketplaces and Mobile Commerce, Online-Marketing
Prof. Dr. J. Marrón Networked Embedded Systems	Sensor Networks, Adaptive Systems, System Software for embedded systems, Data Management in mobile environments, Hoarding / Caching, Ubiquitous/Pervasive Computing, Semi-structured databases
Prof. Dr. K. Pohl Software Systems Engineering	Requirements Engineering, Software Quality Assurance, Software-Architectures, Evaluation of COTS/Open Source-Components
Prof. Dr. Ing. E. Rathgeb Computer Network Technology	Computer Network Technology
Prof. Dr. S. Schneegaß Human Computer Interaction	Mobile, wearable, and ubiquitous computing systems, Implicit Feedback, Usable Security, Smart Clothing, Interaction in Virtual and Augmented Worlds, Ubiquitous Interaction
Prof. Dr. R. Schütte Business Informatics and Integrated Information Systems	Enterprise Systems, IS-Architectures, Digitalization of organisations, Information modelling, Scientific theory problems of the Business Informatics field
Prof. Dr. S. Stieglitz Professional Communication in Electronic Media / Social Media	Digital Enterprise / Digital Innovation, Digital Society

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)

DOI: 10.17185/duepublico/75244
URN: urn:nbn:de:hbz:464-20211228-082939-0